# Algorithmes Heuristiques et Techniques d'Apprentissage - Applications au Probleme de Coloration de Graphe

Daniel Cosmin Porumbel

université angers

Année : 2009
N° d'ordre : 988

# Algorithmes Heuristiques et Techniques d'Apprentissage

# Applications au Problème de Coloration de Graphe

## Thèse de doctorat

Spécialité : Informatique

## École Doctorale STIM, ED 503

Présentée et soutenue publiquement

Le 19 Novembre 2009

À Angers

Par

**Daniel Cosmin PORUMBEL**

Devant le jury ci-dessous :

| | | |
|---|---|---|
| *Président :* | Gérard Plateau, | Professeur à l'Université de Paris-Nord |
| *Rapporteurs :* | Philippe Galinier, | Professeur à l'École Polytechnique de Montréal |
| | Thomas Stützle, | Chercheur Qualifié (Dr. Habil.) au FNRS, Belgique |
| *Examinateur :* | Olivier Hudry, | Professeur à ENST (Télécom ParisTech), Paris |
| *Co-directeur :* | Jin-Kao Hao, | Professeur à l'Université d'Angers |
| *Co-directeur :* | Pascale Kuntz, | Professeur à l'Université de Nantes |

Laboratoire d'Étude et de Recherche en Informatique d'Angers
2 Bd Lavoisier, 49045 Angers Cedex 01

Laboratoire d'Informatique de Nantes Atlantique
2 rue de la Houssinière, 44322 Nantes Cedex 03

# Remerciements

Tout d'abord, je suis très reconnaissant à l'égard de mes encadrants, Jin-Kao Hao et Pascale Kuntz, pour m'avoir fait découvrir un domaine de recherche très intéressant, et pour avoir tout fait afin de me permettre de travailler dans d'excellentes conditions. L'organisation et la planification de cette thèse ont été parfaites - e.g. j'ai passé 1.5 ans à Nantes (équipe COD du LINA) et 1.5 ans à Angers (équipe MOA du LERIA). Je les remercie vivement pour tous les conseils, avis et discussions qui m'ont permis de bien comprendre les idées majeures du domaine de recherche. Cette thèse a été financée par la région Pays de la Loire et je lui en suis très reconnaissant.

Mes sincères remerciements s'adressent à Philipe Galinier pour avoir rapporté ce mémoire et pour son implication dans nos travaux. Je tiens à remercier particulièrement Thomas Stützle pour avoir accepté de rapporter ma thèse et pour avoir apporté des idées et des questions très éclairées et réalistes le jour de la soutenance. Je voudrais remercier aussi particulièrement à Olivier Hudry pour ses observations et questions détaillées qui m'ont permis d'améliorer le manuscrit. Je suis également très reconnaissant envers Gérard Plateau pour son amabilité et son engagement actif dans le bon déroulement du processus de soutenance.

Je profite pour exprimer mes vifs remerciements à tout le collectif d'Angers que j'ai trouvé très flexible et très agréable. En particulier, je remercie Jean-Philippe Hamiez pour avoir partagé avec moi sa riche expertise sur le problème de coloration. Je ne sais pas si Jean-Michel Richer s'est rendu compte que les détails techniques dont nous avons discutés m'ont beaucoup aidé ; je profite pour lui adresser ma profonde gratitude. Je remercie aussi tous les autres enseignant-chercheurs d'Angers qui ont contribué à la relecture du manuscrit et des transparents, notamment Matthieu Basseur, David Genest et Adrien Goëffon. Lionel Chauvin a contribué avec des relectures de détail et je lui en suis très reconnaissant. Je remercie Jorge Maturana et Benoît Da Mota pour leurs nombreuses aides pratiques au cours de temps. Je remercie par la même occasion à tous les autres collègues avec qui j'ai partagé de nombreux bons moments dans les pauses de midi et de café : Sylvain Lamprier, Giglia Gomez, Nadarajen Veerapen, Fabien Chhel, Adila Bouabdallah. J'ai beaucoup apprécié à Angers la qualité du service technique (merci Eric Girardeau, Stéphane Vincendeau et Frantz de Germain) et le secrétariat (merci Catherine Pawlonski). Je remercie aussi Zhipeng Lü pour son aide sur les tests statistiques. Je voudrais rendre hommage à Julien Robet, un doctorant remarquable qui était probablement déjà en mesure de finir sa thèse dans sa deuxième année.

Je remercie mes collègues à Nantes pour les bons moments passés avec eux : Claudia Marinica, Toader Gherasim, Vlad Georgescu, Nicolas Beaume, Pierrick Bruneau et Bruno Pinaud. Merci Julien Blanchard pour les informations concernant les algorithmes de Multidimensional Scaling. Merci Petr Pošík pour le document sur les stratégies évolutionnistes de diversité à base de distances. Je remercie Adina Florea pour son support dans mes projets. Je remercie Dragoş Ghioca pour ses précieux conseils toutes ces années.

Comment aurais-je pu faire passer agréablement les longues journées de travail de recherche sans Daniela ? En dernier point, mais non le moindre, je te remercie, car c'est avec toi que j'ai vécu les plus beaux moments durant la thèse.

Permettez moi de dédier cette thèse à la mémoire du mon grand père Nicolae Marica.

# Table des matières

**Introduction Générale** **1**

**1 Introduction** **5**
1.1 Problèmes difficiles et algorithmes heuristiques . . . . . 6
1.1.1 Techniques d'apprentissage en optimisation combinatoire . . . . . 7
1.1.2 Extraction d'informations "en ligne" . . . . . 11
1.2 Coloration de graphe – plateforme d'évaluation expérimentale . . . . . 12
1.2.1 Domaines d'application . . . . . 13
1.2.2 Définitions formelles et codage de configurations . . . . . 15
1.2.3 Instances DIMACS . . . . . 17
1.2.4 Résultats de référence pour l'évaluation et la comparaison des performances . . . . . 19

**2 RCTS : Un Algorithme Tabou avec de Nouvelles Fonctions d'Évaluation et une Liste Tabou Réactive** **23**
2.1 Introduction . . . . . 24
2.2 Recherche Tabou de base pour la $k$-coloration . . . . . 25
2.2.1 Le voisinage . . . . . 25
2.2.2 Gestion de la liste Tabou . . . . . 26
2.2.3 Spécification de l'algorithme Tabou de base . . . . . 27
2.3 Nouvelles fonctions d'évaluation "bien informées" . . . . . 28
2.3.1 Fonction d'évaluation basée sur le degré . . . . . 28
2.3.2 Des informations dynamiques pour définir d'autres fonctions d'évaluation . . . . . 30
2.3.3 Idées similaires dans la littérature et remarques sur la complexité . . 30
2.4 Une technique réactive pour gérer la liste Tabou . . . . . 31
2.5 Résultats et discussions . . . . . 32
2.5.1 Paramètres . . . . . 33
2.5.2 Influence des fonctions d'évaluation . . . . . 33
2.5.3 Influence de la liste Tabou réactive . . . . . 37
2.5.4 Résultats complets sur toutes les instances difficiles . . . . . 37
2.6 Conclusions . . . . . 39

**3 Cartographie de l'Espace de Recherche** **41**
3.1 Introduction . . . 42
3.1.1 Analyse de l'espace de recherche . . . 42
3.2 La recherche locale typique et la vision globale . . . 43
3.3 Cartographie de l'espace de recherche . . . 45
3.3.1 L'opération de cartographie . . . 45
3.3.2 Distribution spatiale des meilleurs optima . . . 47
3.3.3 Distribution spatiale des configurations visitées en série sur une courte période . . . 48
3.3.4 Distribution spatiale des colorations visitées en série sur une longue période . . . 50
3.4 Conclusion du chapitre . . . 51

**4 Recherches Locales Guidées : TS-Div et TS-Int** **53**
4.1 Introduction . . . 54
4.1.1 Motivation et objectifs . . . 54
4.2 TS–Div : recherche continue de régions inconnues . . . 55
4.2.1 Description formelle de TS–Div . . . 56
4.2.2 Vitesse de TS–Div . . . 59
4.3 TS–Int : un parcours en largeur de l'espace des solutions . . . 62
4.3.1 Cartographie par MDS du parcours TS–Int . . . 63
4.4 Résultats et discussions . . . 65
4.4.1 Procédé expérimental . . . 65
4.4.2 Résultats standards de TS–Div et de TS–Int . . . 66
4.4.3 TS–Int – localisation exacte d'une solution à partir d'une localisation approximative . . . 68
4.4.4 La structuration des graphes et du paysage de recherche . . . 68
4.4.5 Temps d'exécution de TS–Div et de TS–Int . . . 70
4.5 Vers des applications à d'autres problèmes d'optimisation combinatoire . . . 70
4.6 Conclusions de chapitre . . . 72

**5 Contrôle de Diversité et Croisement Informé dans L'Approche Évolutionniste** **75**
5.1 Introduction . . . 76
5.2 Présentation générale de l'algorithme évolutionniste . . . 77
5.2.1 Terminologie du calcul évolutionniste . . . 77
5.2.2 L'algorithme mémétique . . . 78
5.3 Croisement "bien informé" . . . 79
5.3.1 Les meilleures caractéristiques héritables et le nombre de parents . . . 80
5.3.2 Travaux connexes . . . 82
5.4 Préservation et création continue de diversité . . . 82
5.4.1 Distance entre les individus . . . 83
5.4.2 Rejet des enfants . . . 83
5.4.3 Stratégie de remplacement . . . 86

5.4.4 Idées similaires dans la littérature . . . . . 88
5.5 Résultats et discussions . . . . . 90
5.5.1 Conditions expérimentales . . . . . 90
5.5.2 Résultats avec un temps standard de 300 minutes . . . . . 91
5.5.3 Comparaison avec les meilleurs algorithmes . . . . . 94
5.5.4 Influence de la diversité, du croisement, et de la fonction d'évaluation 95
5.6 Conclusions . . . . . 97

**6 Distance de Transfert : Calcul Rapide et Interprétation Spécifique à la Coloration 99**
6.1 Introduction . . . . . 100
6.1.1 Contexte et travaux connexes . . . . . 100
6.1.2 Pourquoi un nouvel algorithme ? . . . . . 101
6.2 Définition formelle de la distance . . . . . 102
6.3 Calcul de la distance . . . . . 103
6.3.1 Matrice de similarité $T$ en $O(|S|)$ . . . . . 104
6.3.2 Affectation maximale en $O(|S|)$ . . . . . 105
6.4 Extensions . . . . . 108
6.5 Une étude de cas pour la coloration de graphe . . . . . 109
6.6 Remarques finales . . . . . 110
6.6.1 Pseudocode de l'algorithme . . . . . 110

**Conclusion Générale 113**

**Liste des figures, tableaux et algorithmes 117**

**Références bibliographiques 121**

**Publications 134**

**Résumé / Abstract 138**

# Introduction Générale

## Contexte du travail

Les problématiques étudiées dans cette thèse portent sur l'optimisation combinatoire et sur les stratégies heuristiques de résolution. Un des écueils bien connus des algorithmes heuristiques est leur difficulté à sortir de bassins d'attraction d'optima locaux de qualité plus mauvaise que celle de l'optimum global. Plus généralement, nous observons que les stratégies heuristiques classiques manquent parfois d'une vue d'ensemble ; il semble plutôt difficile d'intégrer une "auto-conscience" dans un processus de résolution. Pour illustration, certaines recherches locales peuvent être comparées à la « recherche du mont Everest dans un brouillard épais tout en souffrant de l'amnésie » [Russell and Norvig, 2002].

Cette thèse est consacrée à l'élaboration de stratégies pour faire face à ce type de difficultés. Par exemple, il pourrait être possible de réduire les effets de type "amnésie" en collectant des informations pertinentes pendant le processus de résolution, afin de guider plus efficacement l'exploration de l'espace de recherche. Une piste de recherche en plein essor consiste à combiner des heuristiques de résolution avec des méthodes de fouille de données et d'apprentissage. L'objectif de la fouille de données concerne « l'extraction d'informations implicites, auparavant inconnues, et potentiellement utiles » [Frawley *et al.*, 1992] à partir de grands volumes de données. Cet objectif n'est pas si éloigné de celui du chercheur en optimisation combinatoire confronté au déploiement de méthodes d'exploration dans de très vastes espaces de recherche. Dans cette thèse notre cadre expérimental a été celui du problème de la coloration de graphes.

## Objectifs

L'un de nos objectifs principaux a été de concevoir des techniques pour rendre la stratégie de recherche plus "informée", plus "auto-consciente". Bien que l'apprentissage automatique semble une bonne piste, il demeure un certain nombre de challenges à relever. Le plus important concerne le niveau opérationnel et la complexité de calcul : il est essentiel que toute information supplémentaire intégrée dans un processus de résolution n'introduise pas d'importants coûts supplémentaires de calcul.

Le cadre conceptuel pour atteindre l'objectif ci-dessus est fondé sur une mesure de distance qui permet de comparer les solutions candidates (configurations). En analyse de données, la détection de structures met généralement en œuvre une comparaison : par exemple, lorsque l'on classifie des données, on cherche à regrouper dans une même classe les individus qui se ressemblent et dans des classes séparées des individus qui diffèrent. D'un point de vue opérationnel, cette comparaison est basée sur le concept de distance. Un challenge particulier pour le problème de coloration a été de concevoir un algorithme très rapide pour calculer cette distance.

Un objectif plus général a été d'établir des principes de guidage et d'orientation dans l'espace de recherche en faisant appel à cette mesure de distance. Par exemple, pour avoir une vue d'ensemble de la trajectoire d'une recherche locale, il suffit d'enregistrer un ensemble restreint de sphères – une sphère couvre toutes les configurations jusqu'à une certaine distance d'une "configuration centre". Nous avons cherché à utiliser des principes d'orientation dans l'espace de recherche pour concevoir à la fois des recherches locales et des algorithmes évolutionnistes. Pour tenter d'évaluer sous différents angles l'apport de connaissances supplémentaires dans nos heuristiques, nous avons considéré la possibilité d'introduire de nouvelles fonctions d'évaluation dans un algorithme Tabou.

Les algorithmes que nous avons développés ont été appliqués au cadre de la coloration mais pourraient être transposés à d'autres problèmes d'optimisation, pourvu qu'il soit possible de définir une distance entre les solutions candidates du problème.

## Principales contributions

Notre apport principal concerne quatre directions majeures :

1. l'analyse de l'espace de recherche (cartographie) permettant de dégager des propriétés structurelles sur les solutions potentielles, afin de guider des stratégies de recherche locale ;
2. le développement d'un nouvel algorithme évolutionniste avec un croisement "informé" et une diversité de population garantie par la mesure de distance entre les individus ;
3. l'introduction de fonctions d'évaluation plus informées, spécifiques au problème traité ;
4. un algorithme exact pour calculer la distance de transfert entre partitions/colorations.

**Analyse d'Espace et Recherches Locales Guidées :** Nous avons analysé la distribution spatiale des configurations de grande qualité obtenues par un algorithme de recherche Tabou. Ces configurations ont été comparées avec une mesure de distance qui représente le nombre minimal de transitions de voisinage pour passer d'une coloration à l'autre. Nous avons eu recours à un algorithme d'analyse de données (ici le *Multidimensional Scaling*) pour établir une "cartographie" de l'espace de recherche. Nous avons montré expérimentalement que ces configurations ne sont pas distribuées aléatoirement, mais qu'elles sont regroupées en classes (clusters) qui peuvent être recouvertes par des sphères de diamètre spécifique.

Cette propriété nous a permis de développer une recherche Tabou (TS–Div) qui enregistre les sphères visitées lors de son exploration et guide ensuite l'exploration vers des sphères non encore visitées. De plus, nous avons ajouté un algorithme d'intensification (TS–Int) qui fait des recherches méticuleuses dans un périmètre limité autour d'une configuration donnée. S'il existe une solution à moins d'une certaine distance de ce point donné, TS–Int la trouve avec un taux de réussite de 100%. L'article complet a été accepté par *Computers & Operations Research* [Porumbel *et al.*, 2010] ; plus de détails et d'idées sur la

partie diversification ont été présentées à *LION 3 (Learning In OptimisatioN)* [Porumbel *et al.*, 2009d].

**Algorithmes Évolutionnistes avec Diversification et Recombinaison Informée :** La mesure de distance a été de nouveau utilisée pour maintenir une dispersion spatiale optimale de la population. Pour illustration, l'algorithme évolutionniste proposé (Evo-Div) insère un nouvel individu dans la population uniquement si il n'est pas trop proche des individus existants. De plus, nous avons développé un croisement "informé", qui exploite les informations les plus pertinentes pour choisir les gènes transférés d'une génération à l'autre. Ce chapitre s'appuie sur un article complet de 25 pages soumis à un journal [Porumbel *et al.*, 2009c], ainsi que sur un article présenté à *EvoCop 2009* [Porumbel *et al.*, 2009a] (un des trois articles sélectionnés pour *best paper award*).

**Fonctions d'Évaluation Informées :** Nous avons tenté d'exploiter les propriétés des instances pour concevoir des fonctions d'évaluation mieux informées. Ainsi, nous avons développé deux fonctions qui permettent de discriminer des configurations initialement considérées comme identiques avec la fonction conventionnelle d'évaluation (nombre de conflits). En introduisant de l'information supplémentaire (structurelle pour une fonction, apprise pour l'autre), une recherche Tabou non sophistiquée devient capable d'atteindre des résultats très performants par rapport à plusieurs algorithmes plus complexes [Porumbel *et al.*, 2007a]. L'influence des fonctions d'évaluation sur une descente pure encore plus basique a été analysée dans un article présenté à *EA'07 (8th International Conference on Artificial Evolution)* [Porumbel *et al.*, 2007b].

**Algorithme Exact de Calcul de la Distance de Transfert entre Colorations :** Plusieurs algorithmes heuristiques que nous avons développés s'appuient sur le calcul d'une distance entre deux colorations. Comme une coloration est ni plus ni moins qu'une partition de l'ensemble des sommets du graphe, on parle de distance de transfert entre deux partitions : elle est définie par le nombre minimum de sommets qui doivent entre transférés entre les classes (de couleurs) d'une partition afin d'obtenir l'autre partition. Ce projet présente un algorithme exact pour le calcul de la distance : si au moins une des conditions présentées est satisfaite, alors il est possible de réduire considérablement (de $O(|V| + k^3)$ à $O(|V|)$) la complexité de la méthode standard – fondée sur l'algorithme hongrois et une réduction directe au problème d'affectation [Porumbel *et al.*, 2009b].

## Plan de la thèse

Le manuscrit est organisé comme suit.

- Dans le premier chapitre, nous décrivons brièvement la portée et les objectifs des algorithmes heuristiques et nous insistons sur les limitations qui sont adressées dans cette thèse. Nous introduisons la coloration de graphe, notre cadre expérimental, et nous établissons la terminologie et les notations de base ;
- Dans le deuxième chapitre, nous abordons un premier algorithme de coloration, une recherche Tabou de base qui servira à plusieurs méthodes de résolution de cette thèse. Nous nous intéressons ici à améliorer cet algorithme basique avec des fonctions d'évaluation bien informées et avec une gestion réactive de la durée Tabou. Cela a

conduit à un algorithme plus évolué et plus efficace ;
- Le troisième chapitre fait une analyse de la distribution spatiale des configurations de grande qualité visitées par une recherche locale définie au chapitre 2. Après un état de l'art sur les méthodes d'analyse de l'espace, nous fournissons des preuves expérimentales pour l'hypothèse de clusterisation : les configurations de grande qualité semblent être relativement proches les unes des autres, et regroupées dans des sphères.
- Le quatrième chapitre présente deux Recherches Locales Guidées en utilisant l'analyse de l'espace du Chapitre 3. TS–Div est orienté vers la diversification et il assure que le processus de recherche ne visite pas les mêmes sphères de façon répétitive. TS–Int est orienté vers l'intensification et assure une exploration complète d'un périmètre limité, en utilisant un parcours en largeur des sphères de ce périmètre.
- Dans le cinquième chapitre, nous présentons l'algorithme évolutionniste Evo-Div qui exploite également l'analyse de l'espace du chapitre 3 afin d'assurer un écart approprié entre les individus.
- Le dernier chapitre présente en détail la distance entre les colorations, et propose un algorithme exact pour son calcul.

Chaque chapitre peut être lu indépendamment, avec un effort minimal de référence à d'autres chapitres. Toutes les contributions sont présentées dans l'ordre chronologique de leur développement, introduisant toutes les notions d'une façon progressive.

# Chapitre 1

# Introduction

Dans la première partie de ce chapitre nous introduisons le contexte et la motivation des approches heuristiques en optimisation combinatoire. Nous portons une attention particulière aux inconvénients des heuristiques qui pourraient potentiellement être améliorés par apprentissage automatique. Dans la deuxième partie, nous présentons notre cadre expérimental – le problème de coloration de graphe ; puis nous décrivons synthétiquement les algorithmes les plus reconnus introduits ces trois dernières décennies.

## Sommaire


**1.1 Problèmes difficiles et algorithmes heuristiques . . . . . 6**
1.1.1 Techniques d'apprentissage en optimisation combinatoire . . . . 7
1.1.2 Extraction d'informations "en ligne" . . . . . 11
**1.2 Coloration de graphe – plateforme d'évaluation expérimentale 12**
1.2.1 Domaines d'application . . . . . 13
1.2.2 Définitions formelles et codage de configurations . . . . . 15
1.2.3 Instances DIMACS . . . . . 17
1.2.4 Résultats de référence pour l'évaluation et la comparaison des performances . . . . . 19

## 1.1 Problèmes difficiles et algorithmes heuristiques

Une *instance* d'un problème d'optimisation combinatoire est définie par un ensemble de solutions candidates (i.e. l'espace de recherche $\Omega$) et une fonction objectif. Résoudre un tel problème (plus précisément, une telle instance du problème) consiste à trouver une solution qui optimise (e.g. qui minimise) la fonction objectif donnée. Dans la pratique, certains problèmes combinatoires sont considérés comme des *problèmes difficiles*, ou non traitables (*intractable problems*). En termes de complexité, ils sont généralement des problèmes $NP$-complets ou $NP$-durs (*NP-hard*) [Garey *et al.*, 1979] : tout algorithme exact de résolution peut nécessiter un temps de calcul prohibitif, qui croît de manière exponentielle avec la taille de l'entrée (nous supposons que $P \neq NP$) [1].

Les problèmes combinatoires difficiles ont fait l'objet de plus de 50 ans de recherche intense dans *plusieurs* communautés liées aux algorithmes, incluant (sans se limiter à celles-ci) : l'algorithmique et la théorie de complexité, l'optimisation, la recherche opérationnelle et l'intelligence artificielle. Des progrès importants ont été réalisés, et ainsi, il existe à ce jour trois approches génériques [Hoos and Stützle, 2004, p. 27] pour tenter de résoudre les problèmes difficiles :

- étudier des classes d'instances qui admettent des algorithmes exacts efficaces ;
- développer des algorithmes d'approximation de temps polynomial ;
- développer des algorithmes heuristiques, métaheuristiques ou stochastiques.

Concernant la première approche, il est important de mentionner qu'un algorithme théoriquement exponentiel peut être remarquablement efficace dans la pratique. L'algorithme du simplexe est un exemple représentatif pour cette situation favorable : bien qu'il ait une complexité exponentielle dans le pire des cas, cet algorithme est couramment employé pour résoudre efficacement un très grand nombre de programmes linéaires. Il arrive que des algorithmes exponentiels résolvent facilement même des instances de grande taille pour certains problèmes pratiques. Cependant, c'est l'inverse qui se produit très souvent, i.e. les algorithmes exacts peuvent avoir des difficultés avec des instances de taille modeste [Hao *et al.*, 1999, §2.3]. C'est aussi le cas de la coloration de graphe : à l'heure actuelle, selon [Malaguti and Toth, in press], on ne connaît pas d'algorithme exact qui puisse colorier des graphes aléatoires avec plus de 80 sommets.

Les problèmes difficiles peuvent être également abordés par des algorithmes polynomiaux d'*approximation* qui *garantissent* une solution avec une qualité bornée par un certain seuil par rapport à la qualité optimale. Bien qu'il y ait des problèmes $NP$-durs pour lesquels des algorithmes d'approximation aient été prouvés (e.g. le problème de bin-packing), la plupart des problèmes $NP$-durs n'admettent pas d'algorithme d'approximation polynômial. En effet, pour la plupart des problèmes les plus difficiles, une solution presque optimale ne peut pas être systématiquement garantie en un temps polynômial.

Une alternative qui a fait ses preuves en pratique pour de nombreux problèmes difficiles

1. La classe de complexité $P$ réunit les problèmes de décision qui peuvent être résolus en temps polynomial sur une machine déterministe. La classe $NP$ fait référence aux problèmes de décision Non-déterministes Polynomiaux qui peuvent être résolus sur une machine non déterministe en temps polynomial. Il est généralement admis que $P \neq NP$. Un problème d'optimisation $X$ est dit $NP$-dur s'il existe un problème de décision $NP$-complet $Y$ tel que $Y$ se réduit à $X$ en temps polynomial [Garey *et al.*, 1979].

de grande taille est l'utilisation d'*algorithmes de recherche heuristique* – appelés simplement *heuristiques* dans cette thèse. Ces algorithmes consomment des ressources limitées de temps et ils peuvent produire des solutions de bonne qualité, mais sans aucune garantie théorique. Les heuristiques permettent des gains très importants au niveau pratique, non seulement pour résoudre des problèmes $NP$-durs classiques, mais aussi pour de nombreux problèmes réels difficiles. Dans la pratique, il n'est pas toujours nécessaire de trouver une solution optimale et des solutions acceptables peuvent être souvent trouvées par recherche heuristique. De plus, les solutions optimales fournies par des algorithmes exacts peuvent être souvent atteintes *plus rapidement* par des heuristiques. Comme indiqué dans [Battiti *et al.*, 2008], « nous sommes condamnés à vivre avec des heuristiques pendant une très longue période, peut-être pour toujours ».

Pour un problème d'optimisation donné, une heuristique est essentiellement un algorithme de "recherche", i.e. qui cherche une solution parmi toutes les solutions potentielles. Comme il est raisonnable de supposer qu'il n'est pas possible d'énumérer toutes les solutions potentielles d'un problème difficile de grande taille [2], la question qui se pose est la suivante : comment peut-on décider du sous-espace de l'espace de recherche le plus propice à explorer en un temps donné ? Pour cela, on peut employer des informations spécifiques au problème, ainsi que des (méta)-stratégies et (méta)-principes de recherche. Dans ce cadre, le but de toute heuristique est d'employer des stratégies de recherche efficaces afin de trouver rapidement les régions les plus prometteuses de l'espace de problème, pour localiser une solution.

Le terme métaheuristique, présenté dans l'article qui a introduit la recherche Tabou [Glover, 1986], représente une classe d'algorithmes heuristiques génériques, qui peuvent être appliqués à tout problème d'optimisation ; la seule condition nécessaire est de définir une représentation des configurations, ainsi qu'une fonction objectif. Certaines métaheuristiques exigent aussi une relation de voisinage (e.g. les recherches locales) ou un opérateur de mutation (les algorithmes évolutionnistes), mais ces composantes sont considérées comme des "boîtes noires". Une métaheuristique n'est pas concernée par leur mise en œuvre précise ; elle s'intéresse uniquement aux (méta) stratégies principales.

### 1.1.1 Techniques d'apprentissage en optimisation combinatoire

Malgré leur manque de comportement théoriquement prouvable, les métaheuristiques ont eu un succès impressionnant pour plusieurs problèmes. [3] Une possibilité pour mieux comprendre le comportement d'un algorithme heuristique consiste à analyser son évolution avec des techniques de statistique, d'apprentissage et de fouille de données. Ces techniques permettent d'extraire des informations globales à partir des grands volumes de données traitées au cours de temps par le processus de recherche – e.g. pour la coloration, un algorithme visite des millions de configurations par minute.

2. Notons qu'une autre approche en optimisation discrète est représentée par les algorithmes à base d'énumération implicite (e.g. branch-and-bound, branch-and-cut). Ces algorithmes sont souvent appliqués lorsque l'espace de recherche peut être énuméré de façon efficace.

3. L'objectif de ce chapitre n'est pas de faire un exposé sur les métaheuristiques ou sur leurs résultats ; nous renvoyons pour cela le lecteur intéressé à [Hoos and Stützle, 2004].

En effet, l'analyse de données permet de « dégager, derrière une grande masse d'informations, des structures d'organisation entre ces dernières » [Lerman *et al.*, 1981]. Cet objectif, présent chez les fondateurs du domaine, a été repris par les pionniers de la fouille de données confrontés au début des années 90 au passage à l'échelle et à l'explosion de la complexité des données à traiter. Ainsi, la fouille de données a pour objectif de fournir des méthodes d'apprentissage automatiques et semi-automatiques pour extraire des connaissances valides et exploitables à partir de grands volumes de données.

L'espace de recherche représente un exemple classique de grand volume de données, et ainsi, ce type de méthode statistique pourrait fournir des informations exploitables, i.e. qui peuvent être utilisées pour rendre la stratégie de recherche "mieux-informée". Certes, le croisement des algorithmes génétiques, la liste Tabou ou la phéromone artificielle dans les algorithmes de fourmis introduisaient déjà une forme d'apprentissage implicite. Toutefois, les méthodes de fouille de données nous permettent d'analyser des données plus complexes, pour aller au-delà des méthodes actuelles de la littérature.

Le recours à l'apprentissage en optimisation semble être une piste de recherche très jeune qui a eu un essor notable au cours de la dernière décennie – de nombreuses références sont disponibles dans [Battiti *et al.*, 2008]. Bien que les objectifs des techniques d'apprentissage puissent varier considérablement par rapport à la communauté d'optimisation, ils ont été déjà classifiés dans quatre approches importantes (voir [Boyan *et al.*, 2000], une riche collection de 14 études groupées en quatre classes). Nous n'affirmons pas que cette classification soit absolue ou sans chevauchement, mais nous la considérons toujours d'actualité pour les directions de recherche les plus actives :

**Analyse de l'espace de recherche** Des informations fondamentales peuvent être extraites par des analyses statistiques des espaces de recherche. En fait, toute heuristique efficace tient compte, explicitement ou implicitement, des propriétés structurelles de l'espace de recherche. Il existe plusieurs approches pour analyser l'espace et elles peuvent se focaliser sur les aspects suivants, sans se limiter à ceux-ci : (i) la forme de la surface de recherche (e.g. convexe ou non convexe, information primordiale dans l'optimisation numérique), (ii) des indicateurs de rugosité et de difficulté (e.g. quelques termes anglo-saxons sont bien représentatifs : *ruggedness, smoothness* ou *fitness-distance correlation*), (iii) les similarités et les variables partagées par les optima locaux (e.g. le *"backbone"*), ou (iv) le nombre, la qualité et la distribution spatiale des optima locaux – voir également [Hoos and Stützle, 2004, Chapitre 5] ou la Section 3.1.1.

**Apprentissage de nouvelles fonctions d'évaluation** Cela est une direction active de recherche, consacrée à des modifications de la fonction objectif qui permettent de mieux guider l'heuristique à travers l'espace des solutions. Pour toute configuration donnée, une meilleure fonction d'évaluation devrait mieux estimer le potentiel pour conduire à une solution. Cela est particulièrement utile si de nombreuses configurations partagent la même valeur de la fonction objectif ; s'il y a une façon de discriminer ces configurations, il y aurait moins d'optima locaux pour le processus de recherche. Bien que la fonction d'évaluation soit souvent conçue par le chercheur, elle peut être aussi ajustée par le processus de recherche en fonction des informa-

tions acquises. L'algorithme *Guided Local Search* [Voudouris and Tsang, 2003] est un exemple représentatif issu de cette direction de recherche ; il propose de modifier la fonction objectif lorsque le processus de recherche est bloqué dans un optimum local.

**Modèles de génération de solutions** Cette direction est consacrée à l'exploitation des informations concernant les optima locaux visités afin de fournir des indications sur la génération de meilleures solutions. Pour cela, on peut créer un modèle pour produire de nouvelles solutions avec des opérateurs spécialisés, e.g. en fixant quelques variables à certaines valeurs (partagées par tous les optima locaux visités), en estimant la distribution des optima globaux, ou en déterminant la "probabilité" de trouver une solution dans une certaine région. Il a été souligné que « les algorithmes génétiques font exactement ce type de modélisation » [Boyan *et al.*, 2000] : la population garde un ensemble de meilleures configurations et les opérateurs génétiques essaient de les combiner pour construire de meilleures solutions. Dans un certain sens, au moins pour certains algorithmes mémétiques en optimisation discrète, cette interprétation des algorithmes génétiques pourrait être plus précise que la théorie Darwiniste. Une présentation pédagogique de *model-driven search* est disponible dans le Chapitre 6 de [Battiti *et al.*, 2008], où il est indiqué que l'optimisation par colonies de fourmis (*Ant Colony Optimization*) et les algorithmes par estimation de distribution (*Estimation of Distribution Algorithms*) sont également fondés sur des principes similaires.

**Sélection d'algorithme et de paramètres** Cette direction a pour objectif d'automatiser le processus de sélection et de configuration de la meilleure heuristique pour un problème – une étape cruciale pour le praticien. Elle porte sur deux aspects principaux :

- **le choix de l'heuristique** Habituellement, le choix de la meilleure heuristique pour un problème est implicitement fait par le chercheur qui utilise son expérience pour apprécier l'efficacité des différents algorithmes – e.g. pour mieux résoudre le problème de SAT, il pourrait préférer une recherche locale plutôt qu'une approche évolutionniste. [4] Cependant, ces décisions peuvent être partiellement automatisées ; de plus, il est possible d'alterner entre plusieurs heuristiques durant le processus d'optimisation. Un exemple représentatif est la classe d'algorithmes *hyper-heuristics* qui sont des heuristiques pour choisir des heuristiques. [5]
- **le choix des paramètres** Aujourd'hui, une technique assez commune consiste à utiliser des "mécanismes réactifs" pour adapter automatiquement les valeurs des paramètres durant le processus de recherche. Le terme "mécanisme réactif" fait référence à une boucle rétroactive qui modifie un paramètre selon l'état actuel de la recherche, d'une manière "en ligne". *Reactive Tabu Search* [Battiti and Tecchiolli, 1994] est une métaheuristique développée autour de ce type de concepts ;

4. Une raison pour cela pourrait être le fait que, les algorithmes évolutionnistes n'ont pas atteint des performances records dans le passé [Hoos and Stützle, 2004, p. 105].

5. Plus de 100 références sont disponibles, en date d'août 2009, dans G. Ochoa's *Bibliography of Hyper-heuristics and Related Approaches*, voir `www.cs.nott.ac.uk/~gxo/hhbibliography.html`

cependant, d'autres algorithmes peuvent avoir des réactions sur le voisinage, sur le programme de recuit, ou sur la fonction d'évaluation – voir Chapitres 2–5 de [Battiti *et al.*, 2008]. Le paramétrage automatique a des implications avec les autres directions de recherche ci-dessus, car un changement de paramètre peut déclencher une transformation complète d'algorithme. Avec un tel changement, on peut transformer un Recuit Simulé en Descente Pure ou en algorithme de Metropolis ; les opérateurs d'algorithmes génétiques peuvent être aussi choisis de cette façon [Maturana, 2009].

La première piste de recherche est entièrement abordée dans le Chapitre 3 dans lequel nous présentons une nouvelle hypothèse de clusterisation pour la coloration de graphe : les configurations de qualité tendent à se trouver relativement proches les unes des autres, groupées dans des sphères de diamètre fixe. Bien qu'il soit difficile de prouver théoriquement une telle hypothèse, nous la soutenons avec des preuves empiriques issues d'expérimentations numériques. On devrait également se rendre compte que d'autres hypothèses peuvent être facilement rejetées pour la coloration de graphe. Par exemple, les hypothèses de type "massif central" ou "grande vallée" [Boese *et al.*, 1994] supposent que les configurations de qualité soient toutes proches de la solution optimale. Cette conjecture semble valide pour le voyageur de commerce, mais, pour la coloration de graphe, nous avons observé que la qualité n'est pas corrélée avec la distance jusqu'à la solution. En fait, une difficulté spécifique au problème de coloration vient du fait que la plupart des solutions "presque optimales" n'ont aucun optimum global dans leur proximité. Nous avons couramment rencontré des instances difficiles qui admettent plusieurs solutions globales très différentes les unes des autres ; il n'y a pas nécessairement de "massif central" dans la coloration *pour les instances difficiles*.

La deuxième direction de recherche concerne les deux nouvelles fonctions d'évaluation introduites au Chapitre 2. Bien que la première fonction soit spécifiée avant d'initialiser l'exploration, la seconde est construite par le processus de recherche en fonction de certaines informations apprises (le nombre de changements de couleurs de chaque variable). L'idée est de donner une importance plus grande aux variables qui changent très souvent d'état (e.g. les sommets qui changent souvent de couleur), afin d'encourager le processus de recherche à fixer leurs valeurs, et ainsi à déclencher plus de diversification – les variables restées fixes plus longtemps dans le passé sont encouragées à changer plus souvent dans la suite.

Le troisième aspect est abordé partiellement par l'algorithme TS–Int de la Section 4.3, un algorithme qui essaie de trouver une solution à partir d'une localisation approximative. Nous avons empiriquement observé que TS–Int peut systématiquement atteindre l'optimum global, s'il est situé à moins d'une certaine distance d'une coloration fournie initialement. En fait, TS–Int explore un périmètre limité autour de *plusieurs optima locaux distants*, fournis par un algorithme de diversification TS–Div : il est très probable que la solution se trouve dans la proximité d'un de ces points.

La dernière approche de recherche est également évoquée dans cette thèse, dans des contextes différents. Deux exemples sont : (i) la durée réactive Tabou présentée dans la Section 2.4 et (ii) la dispersion réactive de l'algorithme évolutionniste Evo–Div (Section

5.4.2.2). Les hyper-heuristiques ne sont pas utilisées dans cette thèse.

### 1.1.2 Extraction d'informations "en ligne"

Les approches d'apprentissage en optimisation peuvent être également classifiées selon que la méthode soit "en ligne" ou "hors ligne". Par exemple, la plupart des analyses de l'espace de recherche font partie des méthodes "hors ligne" : ces analyses sont réalisées dans une étape de pré-optimisation, avant de commencer la phase principale d'optimisation. Bien qu'une analyse préalable puisse fournir des directives essentielles pour le choix et la construction d'un algorithme approprié, elle n'est pas employée pour prendre des décisions *pendant* le processus d'optimisation, e.g. pour réagir à des informations acquises durant l'exploration.

Le réglage réactif des paramètres est un exemple de technique d'apprentissage "en ligne", employée pendant le processus d'optimisation. Une analyse "hors-ligne" de l'espace de recherche a l'avantage de pouvoir recourir à des méthodes statistiques plus complexes (e.g. comme l'algorithme de *Multidimensional Scaling* dans cette thèse), car elle n'a pas de contraintes de temps. En effet, tout apprentissage "en ligne" est limité par de fortes contraintes de complexité, parce qu'il ne doit pas introduire un ralentissement important dans le processus de recherche. Les heuristiques sont habituellement des algorithmes rapides, qui doivent visiter de nombreuses configurations dans un immense espace de recherche. Pour la coloration de graphe, nos algorithmes peuvent visiter quelques *millions de configurations par minute*. Dans la partie d'analyse de l'espace, nous utilisons des méthodes d'analyse de données qui traitent que quelques milliers de configurations par heure.

Pour ces raisons, les techniques d'apprentissage en ligne les plus communes sont habituellement très "légères" (peu coûteuses en temps de calcul), souvent à base de réactions à des événements locaux ; par exemple, le réglage réactif de la durée Tabou (e.g. voir Section 2.4) est fondé sur des informations d'histoire courte. Il semble qu'il soit plutôt problématique d'extraire et d'intégrer "en marche" (*"on the fly"*) une grande quantité d'informations globales, e.g. des régles d'association entre la localisation et la qualité des configurations visitées, des modèles de distribution d'optima locaux, des propriétés de surface et paysage de l'espace, etc.

Cependant, un objectif de cette thèse est de montrer qu'une procédure d'apprentissage très "légère" est suffisante pour réaliser des tâches apparemment complexes, e.g. enregistrer le chemin complet d'exploration d'un processus d'optimisation. L'idée principale est que, bien que l'ensemble total de configurations visitées soit trop grand pour pouvoir les stocker, un enregistrement à grain grossier peut être réalisé en enregistrant un plus petit nombre de *sphères*. La notion de sphère est ici définie en utilisant une mesure de distance entre les configurations de l'espace de recherche : une sphère est l'ensemble de toutes les configurations situées à moins d'une certaine distance d'une configuration centrale.

La Figure 1.1 montre un exemple d'un enregistrement à gros grain. Une fois équipée avec une telle composante d'enregistrement, toute recherche locale peut "détecter" le moment où elle entre dans des régions (sphères) déjà visitées. Ainsi, la recherche peut devenir "auto-informée", i.e. elle peut savoir si elle est toujours attirée par certaines régions (visitées à de nombreuses reprises), ou si elle assure la diversification globale ou pas.

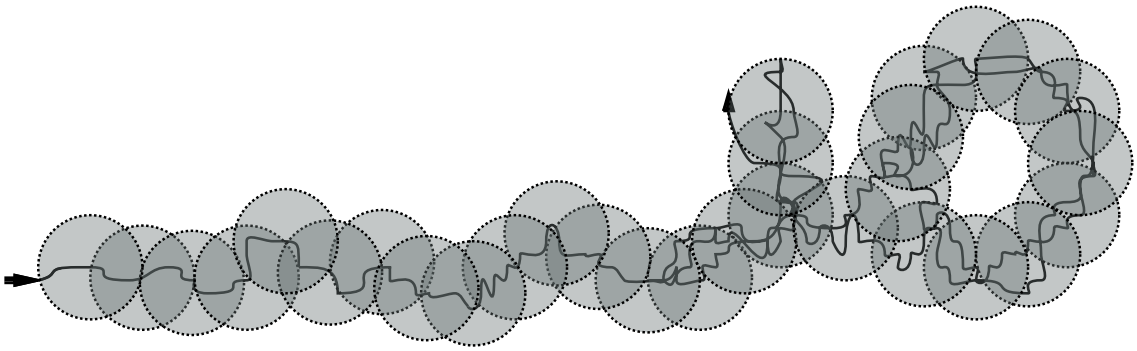

FIGURE 1.1 – Exemple d'un enregistrement à gros grains du chemin d'exploration d'une recherche locale. Le nombre de sphères est beaucoup plus petit que le nombre total de points de l'espace de recherche visités. Cet enregistrement nécessite seulement *un calcul de distance par itération.*

Ce type de composante d'enregistrement peut être attaché à toute recherche locale si une mesure de distance peut être calculée efficacement dans l'espace de recherche. De plus, cette composante est peu coûteuse en temps de calcul : elle doit seulement calculer *une distance par itération.* Plus exactement, à chaque itération, elle calcule la distance jusqu'au centre de la sphère courante : si le processus de recherche est toujours à l'intérieur de la sphère courante, aucune action n'est effectuée ; sinon, la configuration courante devient un centre pour la prochaine sphère courante. On pourrait avoir besoin de plus de calculs de distances *uniquement lorsque* l'algorithme doit utiliser cet enregistrement pour prendre des décisions calculées. Cependant, certaines actions importantes (e.g. déclenchement de diversification) peuvent être décidées avec une quantité raisonnable de calcul supplémentaire, en maintenant le nombre de calculs de distances dans des limites acceptables (voir, par exemple, Section 4.2.2).

## 1.2 Coloration de graphe – plateforme d'évaluation expérimentale

Cette section décrit le problème d'optimisation considéré dans cette thèse, les instances standards, ainsi que les approches précédentes qui nous servent d'étalon pour l'évaluation de nos nouveaux algorithmes.

La coloration de graphe est probablement l'un des problèmes d'optimisation combinatoire les plus étudiés en informatique et en mathématiques. Il peut être formulé d'une façon très simple : colorier les sommets d'un graphe avec le nombre minimum de couleurs (le nombre chromatique) tel qu'il n'existe pas deux sommets adjacents de la même couleur.

D'un point de vue complexité théorique, la coloration de graphe a été l'un des premiers problèmes démontrés $NP$-complet [Karp, 1972]. En effet, sa difficulté est bien connue : hormis pour un nombre très limité de cas spéciaux (arbres, graphes bipartis, etc.), le

nombre chromatique ne peut même pas être approximé en temps polynômial dans un facteur constant par rapport à l'optimum (sauf si $P = NP$ !). Une preuve pour un facteur de 2 est donnée depuis [Garey *et al.*, 1979, Théorème 6.11], mais de nouveaux résultats sur d'autres facteurs sont également disponibles [Lund and Yannakakis, 1994; Bellare *et al.*, 1998].

La littérature sur la coloration de graphe est très vaste, et ainsi, nous insistons ici uniquement sur les publications ayant de forts rapports avec cette thèse. En effet, pendant les cinq dernières années, un nombre considérable de monographies, d'articles de synthèse et de numéros spéciaux ont été publiés dans des journaux majeurs du domaine [Galinier and Hertz, 2006; Chiarandini *et al.*, 2007; Malaguti and Toth, in press; Johnson *et al.*, 2008] avec des références à des centaines de publications – on peut dire, sans exagérer, des milliers de publications. [6] De plus, pendant la dernière décennie, j'ai relevé au moins sept thèses ayant une partie majeure consacrée à la coloration heuristique de graphe [Galinier, 1999; Dorne, 1998; Blöchliger, 2005; Zufferey, 2002; Chiarandini, 2005; Devarenne, 2007; Weinberg, 2004].

### 1.2.1 Domaines d'application

Le problème de coloration de graphe est lié à de nombreuses applications répandues dans des domaines variés, tels que :

- l'affectation de fréquences dans les réseaux cellulaires [Hale, 1980; Gamst and Rave, 1982; De Werra and Gay, 1994; Dorne and Hao, 1995; Park and Lee, 1996] ;
- l'ordonnancement [Leighton, 1979; Gamache *et al.*, 2007; Lewandowski and Condon, 1996; Zufferey *et al.*, 2008; Laurent and Hao, 2009] ;
- l'affectation des registres dans les compilateurs [Chaitin, 1982; Lewandowski and Condon, 1996; Chow and Hennessy, 1990] ;
- les emplois du temps [de Werra, 1985; Burke *et al.*, 1994] ;
- la gestion de chaînes logistiques [Lim and Wang, 2005; Glass, 2002] ;
- le calcul de dérivées, de matrices jacobienne et hessiennes [Gebremedhin *et al.*, 2005] ;
- la gestion du trafic aérien [Barnier and Brisset, 2004] ;
- l'allocalisation de ressources en réseau [Woo *et al.*, 1991] ;

En fait, une liste complète d'applications (directes ou implicites) aurait probablement des centaines de références. La liste ci-dessus comprend *uniquement les applications les plus connues*. En effet, des exemples d'applications bien plus spécifiques sont disponibles dans la littérature (e.g. la vérification des circuits électroniques, le triage de trains, la manufacture) et nous renvoyons pour cela le lecteur intéressé à l'introduction de [Malaguti *et al.*, 2008; Lü and Hao, 2010; Blöchliger and Zufferey, 2008; Malaguti and Toth, in press] ; voir aussi une quinzaine de références dans [Johnson and Trick, 1996, p. 2]. Dans un certain sens, la coloration de graphe est même très connue pour le grand public par l'intermédiaire du Sudoku – ce jeu est ni-plus-ni-moins qu'un problème de coloration avec $k = 9$ couleurs.

6. La bibliographie de M. Chiarandini's (`www.imada.sdu.dk/~marco/gcp/`) contient une collection très riche, régulièrement réactualisée – environ 200 références, en date de juin 2009. Il est intéressant de voir aussi la *Graph Coloring Bibliography* de J. Culberson (`http://web.cs.ualberta.ca/~joe/Coloring/index.html\#Color.bibliography`) qui comprend presque 500 références.

#### 1.2.1.1 Approches heuristiques existantes

Les premiers algorithmes pour la coloration de graphe ont été développés dans les années soixante [Welsh and Powell, 1967; Christofides, 1971; Brown, 1972]. Depuis lors, un nombre considérable de nouvelles techniques ont été développées et d'importants progrès ont été enregistrés. Tandis que des approches d'optimisation combinatoire à base d'énumération complète implicite (e.g. branch and bound) ont été testées pour la coloration exacte [Brelaz, 1979; Mehrotra and Trick, 1996; Sewell, 1996; Ramani *et al.*, 2006; Lucet *et al.*, 2006; Méndez-Díaz and Zabala, 2006], « très peu d'algorithmes exacts sont disponible pour le problème » [Méndez-Díaz and Zabala, 2006]. Les algorithmes exacts peuvent actuellement résoudre uniquement des graphes aléatoires de petite taille, avec jusqu'à 80 sommets [Malaguti and Toth, in press].

Ainsi, les méthodes heuristiques dominent la littérature pour des problèmes de coloration *générique*[7] sur des graphes ayant plus de quelques centaines de sommets. Les algorithmes heuristiques appartiennent essentiellement à trois approches de résolution :

1. Construction séquentielle – e.g. Dsatur [Brelaz, 1979], Iterated Greedy [Culberson and Luo, 1996], RLX and XRLF [Johnson *et al.*, 1991], méthodes très rapides mais pas particulièrement efficaces ;
2. Recherches locales
   - Recherche Tabou [Hertz and Werra, 1987; Fleurent and Ferland, 1996a; Dorne and Hao, 1998b; Galinier and Hao, 1999; Blöchliger and Zufferey, 2008] ;
   - Recuit simulé [Chams *et al.*, 1987; Johnson *et al.*, 1991] ;
   - Recherche à voisinage variable ou espace de recherche variable [Trick and Yildiz, 2007; Avanthay *et al.*, 2003; Hertz *et al.*, 2008]) ;
   - Recherche locale itérée [Paquete and Stützle, 2002; Chiarandini and Stützle, 2002; Lü and Hao, 2009] ;
3. Méthodes à basé de populations ou distribuées
   - Algorithmes hybrides évolutionnistes [Fleurent and Ferland, 1996a; Dorne and Hao, 1998a; Galinier and Hao, 1999; Galinier *et al.*, 2008; Malaguti *et al.*, 2008; Malaguti and Toth, 2008; Lü and Hao, 2010]
   - Algorithme de colonies de fourmis [Costa and Hertz, 1997; Shawe-Taylor and Žerovnik, 2001; Plumettaz *et al.*, in press]
   - Algorithme hybride distribué [Morgenstern, 1996]

Dans cette vaste littérature, plusieurs travaux de recherche sont particulièrement importants pour notre étude. Un premier pas important dans la coloration pratique a été l'algorithme Tabucol (recherche TABU pour la COLoration) présenté par Hertz et Werra il y a plus de vingt ans [Hertz and Werra, 1987]. Une importante amélioration ultérieure de Tabucol a été représentée par une technique incrémentale qui a permis d'examiner plus rapidement le voisinage [Fleurent and Ferland, 1996a]. Ensuite, d'autres développements importants ont été consacrés à la gestion de la durée Tabou, afin de décider combien

7. Notons qu'une direction de recherche différente est centrée sur l'étude des algorithmes exacts pour des classes *particulières* de graphes, des types *particuliers* de colorations, ou d'autres problèmes reliés qui ne sont pas $NP$-complets.

de temps il fallait interdire un mouvement après son exécution [Dorne and Hao, 1998b; Dorne and Hao, 1998a; Galinier and Hao, 1999; Blöchliger and Zufferey, 2008; Devarenne *et al.*, 2006]. La chronologie des développements de Tabucol est présentée plus en détail dans un article de synthèse [Galinier and Hertz, 2006] ; voir également la Section 2.2.

Des progrès essentiels dans la coloration ont été réalisés grâce aux premiers modèles de croisements fondés sur des groupes (classes), au lieu des couleurs [Dorne and Hao, 1998a; Galinier and Hao, 1999]. Ces croisements ont permis d'améliorer les meilleurs résultats pour plusieurs graphes difficiles de l'époque. Depuis leur intégration dans les algorithmes mémétiques (méthodes évolutionnistes incorporant une recherche locale), les algorithmes à base de populations sont considérés comme une des meilleures approches pour la coloration de graphe.

Notons toutefois que, très récemment, deux algorithmes de recherche locale moins traditionnels ont été très compétitifs [Hertz *et al.*, 2008; Blöchliger and Zufferey, 2008]. Ils ont introduit des structures plus avancées d'espace de recherche et de voisinage, ainsi qu'un codage sous forme de colorations légales partielles, avec un réglage réactif de la liste Tabou. Il est utile de mentionner aussi un article plus ancien [Morgenstern, 1996] qui a joué un rôle important dans la communauté de coloration pour plusieurs raisons. Cet article a présenté de nombreuses idées, certaines utilisées jusqu'à aujourd'hui ; de plus, il a aussi introduit plusieurs algorithmes et les résultats globaux – la plupart d'entre eux récapitulés à la page 357 [Morgenstern, 1996] – constituent toujours un point de référence pour d'autres algorithmes.

Concernant l'évaluation expérimentale des algorithmes, le deuxième challenge DIMACS [Johnson and Trick, 1996] a rassemblé un grand ensemble de graphes qui est devenu un jeu de tests standard pour les algorithmes de colorations depuis 1996. Ce challenge, ainsi que d'autres évènements internationaux récents [Johnson *et al.*, 2002; Johnson *et al.*, 2008] ont intensifié la recherche dans ce domaine très concurrentiel. En effet, cinq des meilleurs algorithmes de la littérature ont été publiés pendant les derniers 18 mois [Malaguti *et al.*, 2008; Hertz *et al.*, 2008; Blöchliger and Zufferey, 2008; Galinier *et al.*, 2008; Lü and Hao, 2010].

### 1.2.2 Définitions formelles et codage de configurations

#### 1.2.2.1 Coloration et $k$-coloration

Soit $G_{V,E}$ un graphe connexe avec un ensemble $V$ de sommets et un ensemble $E$ d'arêtes. Le *problème de coloration,* ou le *problème du nombre chromatique,*[8] consiste à trouver le nombre chromatique de $G_{V,E}$, i.e. le plus petit nombre $k$ tel que le problème de $k$-coloration associé à $G_{V,E}$ et $k$ ait une solution. La $k$-coloration est un problème fortement lié au problème de coloration, défini formellement comme suit :

**Definition 1.1.** *($k$-coloration) : Étant donné un graphe connexe $G_{V,E}$ et $k$ couleurs représentées par des nombres $\{1, 2, \ldots, k\}$, décider s'il existe une $k$-coloration (une affectation*

8. Ou aussi "minimum coloring problem". Notons que si le graphe n'est pas connexe, le problème revient à déterminer le nombre chromatique maximum d'une composante connexe.

*de ces k couleurs aux sommets) sans* conflit*, i.e. sans arête avec les deux extrémités de la même couleur.*

La plupart des algorithmes d'optimisation considèrent le problème général de coloration comme une série de problèmes de $k$-coloration de plus en plus difficiles. Cette méthode commence par fixer une valeur initiale $k$ très grande (e.g. $k = |V|$), puis décrémente itérativement $k$ après avoir trouvé une $k$-coloration sans conflit. Le problème de $k$-coloration devient de plus en plus difficile jusqu'à ce que l'algorithme ne puisse plus trouver une $k$-coloration sans conflit. Le $k$ le plus petit pour lequel le problème de $k$-coloration a été résolu constitue la meilleure solution pour le problème général de coloration ; il est une borne supérieure pour le nombre chromatique $\chi_G$.

#### 1.2.2.2 Représentation et fonction objectif pour la $k$-coloration

**Definition 1.2. Représentation à base de couleurs** *Étant donné un graphe $G_{V,E}$ et $k \in \mathbb{N}^*$, une coloration est une fonction $C : V \rightarrow \{1, 2, \cdots, k\}$, codée comme un tableau de couleurs $C = [C(1), C(2), \ldots, C(|V|)]$.*

L'ensemble de toutes les colorations (solutions candidates, solutions potentielles, configurations) représente l'espace de recherche $\Omega$ de *l'instance de k-coloration* $(G, k)$. Une coloration $C$ est une coloration légale si et seulement si $C(i) \neq C(j)$, $\forall \{i, j\} \in E$. Une coloration légale représente une *solution* pour une instance $(G, k)$. Bien que nous utilisions ce codage dans nos programmes, il est très utile d'interpréter une coloration comme une partition de l'ensemble $V$.

**Definition 1.3. Représentation à base de partitions** *Une k-coloration $C = [C(1), C(2), \ldots, C(|V|)]$ de $G$ est une partition $\{C^1, C^2, \ldots, C^k\}$ de $V$ (i.e. un ensemble de k sous-ensembles disjoints de $V$ et qui forment un recouvrement de $V$) telle que $\forall x \in V$, $x \in C^i \Leftrightarrow C(x) = i$.*

Nous disons que $C^i$ est la classe $i$ de couleurs, induite par la coloration $C$, i.e. l'ensemble de sommets ayant la couleur $i$ dans $C$. La coloration $C$ est une coloration légale ou sans conflit (une solution) si et seulement si aucune classe de couleur de $C$ ne contient des sommets adjacents. Cette définition à base de partitions est particulièrement utile pour éviter des problèmes de symétrie résultant du codage classique à base de couleurs. Ainsi, la représentation à base de partitions est employée dans plusieurs parties de la thèse, e.g. pour définir une distance pertinente entre les colorations (voir Chapitre 6) ou pour définir l'opérateur de croisement dans l'approche évolutionniste (Section 5.3).

**Definition 1.4. Conflits** *Étant donnée une k-coloration $C$, l'ensemble des* conflits *$CE(C)$ définit les arêtes en conflit, i.e. $CE(C) = \{\{i, j\} \in E \ : \ C(i) = C(j)\}$. L'ensemble de sommets en conflit induits par $C$ est $CV(C) = \{i \in V \ : \ \exists j \in V$ tel que $\{i, j\} \in CE\}$.*

S'il n'y pas de risque de confusion, il est possible d'omettre l'argument $(C)$ et de noter $CE$ et $CV$ l'ensemble des arêtes en conflit et, respectivement, des sommets en conflit.

**Definition 1.5. Fonction objectif pour la $k$-coloration** *Étant donné un problème de $k$-coloration $(G_{V,E}, k)$, la fonction objectif $f$ affecte à chaque coloration le nombre de conflits $|CE|$. Ainsi, $f$ est définie par la formule :*

$$f(C) = |\{\{i,j\} \in E : C(i) = C(j)\}|\,,\ \forall C \in \Omega \tag{1.1}$$

Le problème de $k$-coloration est un problème de décision (déterminer s'il existe ou pas une $k$-coloration légale) qui est résolu comme un problème d'optimisation : trouver la valeur minimum de $f(C)$ parmi toutes les colorations $C$ de l'espace de recherche $\Omega$. Une instance de $k$-coloration $(G, k)$ est considérée résolue si et seulement si l'on trouve une coloration telle que $f(C) = 0$. Dans cette thèse, nous n'utilisons pas d'algorithmes capables de faire une énumération complète de l'espace pour démontrer des propriétés telle que $f(C) > 0, \forall C \in \Omega$ ; ce type d'approche est réservé aux instances de petite taille. En effet, le nombre chromatique est toujours inconnu pour plusieurs graphes aléatoires (avec $|V|$ entre 125 et 1000) introduits depuis une vingtaine d'années – i.e. les graphes *dsjc* [Johnson *et al.*, 1991], voir Section 1.2.3 ci-dessous.

Ainsi, cette thèse traite directement le problème de $k$-coloration, i.e. étant donnée une instance $(G_{V,E}, k)$, notre objectif consiste à minimiser la fonction objectif $f$. En optimisation combinatoire, une instance $(G_{V,E}, k)$ représente un cas particulier (une instance) d'un *problème de satisfaction de contraintes* classique, avec $|V|$ variables entières et $|E|$ contraintes de type *toutDifferent* appliquées sur certaines paires de variables.

Un changement de couleur d'un sommet en conflit représente un *mouvement*, ou une transition de voisinage dans une recherche locale – voir aussi Section 2.2.1. Un *optimum local* est une configuration telle qu'il n'existe pas de mouvement qui pourrait conduire à une meilleure valeur de la fonction objectif. Deux notions supplémentaires utiles pour comprendre la structure des espaces de recherche sont le *plateau* et le *bassin d'attraction*. Un plateau est un ensemble de configurations avec la même valeur de fonction objectif, telles que n'importe quelles deux configurations peuvent être reliées par des mouvements à l'intérieur du plateau. Le bassin d'attraction d'un minimum local $C$ peut être défini comme l'ensemble de configurations $C'$ qui peuvent être reliées à l'optimum local $C$ sans passer par des configurations de qualité inférieure à celle de $C'$ – voir [Hoos and Stützle, 2004, §5] pour des définition formelles.

Enfin, notons que dans la partie évolutionniste (Chapitre 5), quelques concepts sont référencés selon une terminologie différente, en conformité avec la tradition de cette communauté. Pour illustration, tous les chercheurs en calcul évolutionniste utilisent "individu" au lieu de "solution potentielle", "solution candidate" ou "configuration" ; néanmoins, tous ces termes ont la même signification.

### 1.2.3 Instances DIMACS

Les graphes standards utilisés pour mener des comparions expérimentales dans la coloration sont ceux du deuxième "challenge" DIMACS [Johnson and Trick, 1996]. Cet événement a collecté 47 graphes, issus de plusieurs familles :

1. graphes aléatoires classiques *dsjc*X.Y avec X sommets et de densité indiquée par Y [Johnson *et al.*, 1991] ;

2. graphes flat $flat$X.Y, générés en partitionnant l'ensemble des sommets en $k_p$ classes et en ajoutant des arêtes uniquement entre les sommets de classes différents (X= $|V|$ et Y est le nombre chromatique $k_p$) [Culberson and Luo, 1996] ;
3. graphes de Leighton $le$X.Y avec X=450 sommets et avec un nombre chromatique connu Y (ils ont une clique de taille Y) [Leighton, 1979] ;
4. deux familles de graphes géométriques aléatoires ($r$X.Y et $dsjr$X.Y) construits à partir d'un ensemble de $X$ points aléatoires (dans le carré unité) en joignant chaque couple de points distants de moins d'un seuil de longueur (indiqué par $Y$). Les graphes $dsjr$ ont été introduits dans le même article que les graphes aléatoires $dsjc$, voir [Johnson *et al.*, 1991] – le préfix $dsj$ vient de David S. Johnson. La famille $r$ a été construite par M. Trick avec un programme de C. Morgenstern, voir également `mat.gsia.cmu.edu/COLOR/instances.html`. Pour ces graphes, il est possible d'ajouter un suffixe "c" pour indiquer le complémentaire du graphe ;
5. Graphes d'emplois du temps ($school1$, $school1.nsh$) et un graphe de carré latin ($latin\_square\_10$) [Lewandowski and Condon, 1996] ;
6. Graphes aléatoires énormes ($C2000.5$ et $C4000.5$) avec 2000 et respectivement 4000 sommets et jusqu'à 4 millions d'arêtes [Lewandowski and Condon, 1996].

#### 1.2.3.1 Bornes supérieures et instances faciles

Le Tableau 1.1 montre des instances DIMACS de $k$-coloration qui ont été résolues facilement par *toutes* nos heuristiques ; une solution a toujours été atteinte dans un temps restreint (secondes ou quelques minutes). Plusieurs d'autres algorithmes ont trouvé des colorations légales avec le même nombre de couleurs $k^*$, mais il n'y a aucune mention d'une coloration légale avec moins de couleurs – i.e. en utilisant $k^* - 1$ couleurs, l'instance devient *probablement unSAT*, ou incoloriable. Dans ce qui suit, nous nous restreignons aux cas restants, considérés dans la littérature comme des *instances difficiles*.

| $G$ | $k^*$ | $G$ | $k^*$ | $G$ | $k^*$ | $G$ | $k^*$ |
|---|---|---|---|---|---|---|---|
| $dsjc125.1$ | 5 | $r125.1$ | 5 | $le450.5a$ | 5 | $flat300.20$ | 20 |
| $dsjc125.5$ | 17 | $r125.5$ | 36 | $le450.5b$ | 5 | $flat300.26$ | 26 |
| $dsjc125.9$ | 44 | $r125.1c$ | 46 | $le450.5c$ | 5 | $flat1000.50$ | 50 |
| $dsjc250.1$ | 8 | $r250.1$ | 8 | $le450.5d$ | 5 | $flat1000.60$ | 60 |
| $dsjc250.5$ | 28 | $r250.1c$ | 64 | $le450.15a$ | 15 | $school1$ | 14 |
| $dsjc250.9$ | 72 | $r1000.1$ | 20 | $le450.15b$ | 15 | $school1.nsh$ | 14 |
| | | $dsjr500.1$ | 12 | $le450.25a$ | 25 | | |
| | | | | $le450.25b$ | 25 | | |

TABLE 1.1 – Bornes supérieures DIMACS faciles. D'autres articles rapportent exactement les mêmes meilleures bornes supérieures $k^*$ pour ces 27 graphes.

#### 1.2.3.2 Variation de difficulté et interprétation de résultats

Rappelons que le problème général de coloration est résolu par une série de problèmes de $k$-coloration. Pour chaque graphe $G$, tous les algorithmes ont rapporté les résultats sous la forme ($G$, $k$) – e.g. indiquant le problème le plus difficile de $k$-coloration résolu (le plus petit $k$). La difficulté de colorier un graphe est strictement corrélée avec la valeur de $k$. Passer d'un résultat de $k$ à un résultat de $k-1$ n'est pas similaire à décrémenter le nombre de contraintes violées dans un problème de satisfaction de contrainte min-conflits, i.e. *une couleur en moins* n'est pas équivalent avec *une contrainte violée en moins*. Dans notre cas, pour résoudre l'instance avec "une couleur en moins", il faut réduire plusieurs contraintes violées (conflits). Par exemple, un algorithme qui résout une instance $(G, k)$ avec un taux de succès de 100%, pourrait être incapable de trouver une $(k-1)$-coloration avec moins d'une dizaine de conflits (cette situation arrive souvent pour les graphes $flat$).

Pour ces raisons, dans la section *bornes et instances faciles* ci-dessus, nous ne parlons pas de "graphes faciles" ; on parle de graphes faciles uniquement quand un $k$ associé est implicitement référé. Tous les articles de coloration, que nous connaissons, ont toujours rapporté des résultats sous la forme $(G, k)$, comme une liste d'instances résolues. Dans un certain sens, comparer des algorithmes de coloration uniquement de cette manière peut être insuffisant, e.g. si deux algorithmes $A_1$ et $A_2$ ne trouvent pas de solution avec $k$ couleurs, on ne voit pas de différence si $A_1$ atteint 1 conflit et $A_2$ n'atteint jamais moins d'une dizaine de conflits.

### 1.2.4 Résultats de référence pour l'évaluation et la comparaison des performances

Tous nos algorithmes sont évalués sur le benchmark (jeu de tests) DIMACS, par rapport aux résultats provenant de plusieurs approches de coloration – voir la liste ci-dessous. Chaque approche de coloration représente soit un algorithme unique, soit une *classe d'algorithmes* – les résultats sont souvent rapportés en combinant les performances de plusieurs algorithmes ou, au moins, de plusieurs variantes d'un seul algorithme. Nous décrivons très brièvement ci-dessous les meilleures approches, indiquant également une abréviation standard utilisée souvent pour les référencer.

**Algorithmes de recherche locale :**

- MIPS – MInimal-state Processing Search [Funabiki and Higashino, 2000], un algorithme de descente avec des capacités de *hill-climbing*, combiné avec des heuristiques de clique maximale et une étape de construction gloutonne ;
- ILS – Iterated Local Search [Chiarandini and Stützle, 2002; Paquete and Stützle, 2002]), une direction de recherche étudiant deux architectures de recherche locale avec différents opérateurs de perturbation ;
- VNS – Variable Neighborhood Search [Avanthay *et al.*, 2003], un algorithme utilisant plusieurs opérateurs différents de voisinages, qui sont alternés afin d'aider un processus de recherche Tabou à sortir des minima locaux ;
- ALS – Adaptive Local Search [Devarenne *et al.*, 2006], une recherche locale utilisant une technique d'exploration spécifique à un très grand voisinage, avec des mécanismes

de détection de boucle et un nouveau type de liste Tabou ;
- PCOL – Partial Coloring search (ou PartialCol) [Blöchliger and Zufferey, 2008], un algorithme Tabou avec un réglage réactif de la durée Tabou et qui encode les configurations comme de colorations légales partielles (i.e. certains sommets peuvent ne pas être colorés) ;
- VSS – Variable Search Space [Hertz *et al.*, 2008], un algorithme dans lequel le processus de recherche commute itérativement entre plusieurs espaces de recherche, chacun ayant son propre codage, sa propre fonction objectif et son voisinage ;

**Méthodes hybrides et à base de populations :**
- DCNS – Distributed Coloration Neighborhood Search [Morgenstern, 1996], une large collection de techniques de résolution, y compris des méthodes distribuées, des algorithmes Metropolis, et un codage à base de colorations légales partielles ;
- HGA – Hybrid Genetic Algorithms [Fleurent and Ferland, 1996b], une première hybridation d'un algorithme évolutionniste de coloration avec la recherche Tabou, mais avec un croisement traditionnel, orienté vers les couleurs ;
- CISM – Crossover by Independent Sets and Mutation search [Dorne and Hao, 1998a], un algorithme évolutionniste utilisant un croisement fondé sur des ensembles indépendants, en hybridation avec une recherche Tabou ;
- HEA – Hybrid Evolutionary Algorithms [Galinier and Hao, 1999], une approche hybride évolutionniste incorporant Tabucol comme recherche locale, et avec un croisement glouton très efficace à base de partitions ;
- AMCOL – Adaptive Memory Coloring Algorithm (également appelé AmaCol) [Galinier *et al.*, 2008], un algorithme à base de population avec une mémoire centrale contenant des ensembles indépendants qui peuvent être assemblés dans des colorations ;
- MMT – Malaguti-Monaci-Toth algorithm [Malaguti *et al.*, 2008], une combinaison entre l'approche mémétique et la programmation linéaire en nombres entiers (i.e. des techniques à base de génération de colonnes) ;
- MCol – Memetic Algorithm for Graph Coloring (ou MemCol) [Lü and Hao, 2010], un algorithme mémétique très récent avec un opérateur de croisement adaptatif multi-parent et un contrôle efficace de balance diversification/intensification.

Pour tout problème d'optimisation, l'évaluation et la comparaison des résultats sont toujours compliquées par le fait qu'on utilise des machines différentes, des langages de programmation différents, ou des styles d'implémentation différents. Les conditions d'arrêt peuvent également varier de manière significative : tandis que certains articles emploient des indicateurs théoriques indépendants de machine (e.g. une limite du nombre d'itérations), les articles les plus récents imposent une limite de temps, habituellement entre une heure et plusieurs jours – voir des valeurs exactes dans la Section 2.5.4 ou la Section 4.4.5.

Par rapport à d'autres problèmes combinatoires, ces références constituent une base de comparaison à la fois très riche et aussi très d'actualité. Ainsi, tous nos nouveaux algorithmes seront toujours comparés avec les résultats de ces approches – voir les Tableaux 2.3 (p. 40), 4.4 (p. 73) et 5.4 (p. 95). En fournissant le plus d'informations possibles en complément du temps CPU (e.g. le nombre d'itérations), nous essayons toujours de fournir, au moins à titre indicatif, une image très complète des performances de nos algorithmes.

Les résultats rapportés par les articles ci-dessus sont également inclus dans une présentation en ligne des meilleures bornes supérieures pour la coloration de graphe (`http://info.univ-angers.fr/pub/porumbel/graphs/`). Cette présentation a été compilée au cours de cette thèse et contient 20 références, en date de septembre 2009. Une liste complète d'algorithmes heuristiques de coloration contiendrait probablement des douzaines ou des centaines de références, mais on ne peut pas faire des comparaisons avec tous. Certains articles introduisant d'excellents développements algorithmiques ne sont pas listés dans toutes nos comparaisons uniquement pour des raisons très pratiques : soit ils ne sont *pas* focalisés sur les meilleurs résultats pratiques, soit ils ont été acceptés pour publication pendant les derniers mois de cette thèse, quand ce manuscrit était plus qu'à moitié écrit [Prestwich, 2002; Glover *et al.*, 1996; Hamiez and Hao, 2004; Hamiez and Hao, 2001; Lü and Hao, 2009; Bouziri *et al.*, 2008; Fotakis *et al.*, 2001; Lü and Hao, 2010]. Toutefois, notons que toutes les bornes découvertes en 2009 seront indiquées entre crochets dans tous les tableaux de comparaison de résultats (e.g. Tableaux 2.3, 4.4, 5.4).

# Chapitre 2

# RCTS : Un Algorithme Tabou avec de Nouvelles Fonctions d'Évaluation et une Liste Tabou Réactive

Ce chapitre présente une première approche pour le problème de coloration. Il débute par un bref rappel d'un algorithme basique de coloration (Tabucol [Hertz and Werra, 1987]) fondé sur la recherche Tabou [Glover, 1986; Hansen, 1986; Glover and Laguna, 1997]. Encore aujourd'hui, l'algorithme original Tabucol, aussi bien que ses variantes conçues depuis 1987, sont parmi les algorithmes de référence pour la coloration de graphe. Ensuite, nous exposons notre première contribution sur deux nouvelles fonctions d'évaluation qui ajoutent des informations supplémentaires (structurelles ou dynamiques) à la fonction classique du nombre de conflits. Ces nouvelles fonctions d'évaluation permettent à notre algorithme de différencier des configurations non distinguées par la fonction d'évaluation conventionnelle (nombre de sommets en conflits). La deuxième contribution porte sur un simple mécanisme réactif pour ajuster la durée Tabou. Bien que l'algorithme intégrant ces deux nouveaux éléments reste assez simple, il trouve les meilleures solutions pour plusieurs graphes DIMACS difficiles. Le contenu de ce chapitre est soumis pour publication et en cours de révision [Porumbel *et al.*, 2007a].

**Sommaire**


**2.1 Introduction . . . . . 24**
**2.2 Recherche Tabou de base pour la $k$-coloration . . . . . 25**
2.2.1 Le voisinage . . . . . 25
2.2.2 Gestion de la liste Tabou . . . . . 26

2.2.3 Spécification de l'algorithme Tabou de base . . . . . 27
**2.3 Nouvelles fonctions d'évaluation "bien informées" . . . . . 28**
2.3.1 Fonction d'évaluation basée sur le degré . . . . . 28
2.3.2 Des informations dynamiques pour définir d'autres fonctions d'évaluation . . . . . 30
2.3.3 Idées similaires dans la littérature et remarques sur la complexité 30
**2.4 Une technique réactive pour gérer la liste Tabou . . . . . 31**
**2.5 Résultats et discussions . . . . . 32**
2.5.1 Paramètres . . . . . 33
2.5.2 Influence des fonctions d'évaluation . . . . . 33
2.5.3 Influence de la liste Tabou réactive . . . . . 37
2.5.4 Résultats complets sur toutes les instances difficiles . . . . . 37
**2.6 Conclusions . . . . . 39**

## 2.1 Introduction

Plusieurs études antérieures sur la coloration montrent que, parmi les metaheuristiques classiques, les algorithmes fondés sur la recherche Tabou (couramment abrégée TS – Tabou Search) sont parmi les plus simples et les plus efficaces pour colorier des graphes difficiles de grande taille. Notons que plusieurs algorithmes hybrides plus performants [Fleurent and Ferland, 1996a; Dorne and Hao, 1998a; Galinier and Hao, 1999; Galinier *et al.*, 2008; Malaguti *et al.*, 2008; Blöchliger and Zufferey, 2008; Hertz *et al.*, 2008] emploient également un algorithme Tabou comme procédure d'amélioration locale. Nous observons que les algorithmes Tabou existants pour la coloration n'emploient pratiquement pas de technique sophistiquée. Notre premier objectif consiste à améliorer la performance d'un algorithme de base en rendant certaines composantes plus efficaces, tout en conservant la simplicité de l'algorithme.

Dans ce chapitre, nous proposons des améliorations pour deux composantes majeures : la fonction d'évaluation et la gestion de la liste Tabou. En effet, la fonction d'évaluation est essentielle parce qu'elle définit (avec le voisinage) le paysage de recherche et guide le processus d'optimisation. La fonction d'évaluation classique pour la $k$-coloration compte simplement le nombre de conflits, i.e. le nombre d'arêtes avec les deux extrémités de la même couleur. L'objectif de l'algorithme Tabou est de se déplacer pas à pas vers la valeur minimale de cette fonction, pour essayer de trouver une $k$-coloration sans conflit. Cependant, cette fonction d'évaluation ne peut pas différencier de nombreuses $k$-colorations avec le même nombre de conflits, mais avec *un potentiel différent* pour conduire à une solution.

Pour cette raison, nous étudions deux nouvelles fonctions d'évaluation qui emploient des informations supplémentaires. L'objectif de la première fonction est d'intégrer des informations structurales relatives aux degrés des sommets en conflit. La seconde utilise des informations apprises au cours du processus de recherche : les fréquences des changements de couleur. De plus, ce chapitre introduit une procédure réactive simple et efficace pour améliorer la gestion de la liste Tabou. Nous montrons expérimentalement que, tout en

restant raisonnablement simple, l'algorithme résultant (ci-après désigné RCTS – Reinforced Coloring Tabu Search) obtient de bonnes performances sur l'ensemble des graphes DIMACS.

Le reste du chapitre est organisé comme suit. La Section 2.2 introduit les définitions préliminaires ainsi que l'algorithme Tabou de base. Cette description de l'algorithme Tabou est également utile dans d'autres chapitres. La Section 2.3 présente les nouvelles fonctions d'évaluation, et la Section 2.4 introduit la gestion réactive de la liste Tabou. Dans la Section 2.5, nous menons une comparaison expérimentale sur l'ensemble complet des graphes DIMACS, suivie par des conclusions dans la dernière section.

## 2.2 Recherche Tabou de base pour la $k$-coloration

La recherche Tabou a été appliquée pour la première fois au problème de coloration par Hertz et De Werra en 1987, menant à l'algorithme bien connu de Tabucol [Hertz and Werra, 1987]. Tabucol a donné à ce moment-là des résultats remarquables pour des instances très variées. Dans la suite, Tabucol a également inspiré plusieurs nouveaux algorithmes Tabou [Fleurent and Ferland, 1996a; Dorne and Hao, 1998b; Galinier and Hao, 1999] qui ont perfectionné Tabucol pour obtenir de meilleures performances. Une présentation historique de Tabucol, aussi bien qu'une analyse complète des algorithmes Tabou les plus performants, est disponible dans un exposé assez récent [Galinier and Hertz, 2006].

Schématiquement, par rapport à l'algorithme initial Tabucol, les algorithmes Tabou plus récents apportent des améliorations sur deux aspects clés. D'abord, dans [Fleurent and Ferland, 1996a], les auteurs ont introduit une technique très efficace pour pouvoir effectuer une évaluation très rapide du voisinage. Cette technique permet à l'algorithme de sélectionner le meilleur voisin parmi *toutes* les solutions voisines – Tabucol utilisait seulement un échantillon du voisinage complet. Ensuite, plusieurs techniques plus raffinées pour la gestion de la liste Tabou ont été développées. Par exemple, dans [Dorne and Hao, 1998b; Dorne and Hao, 1998a; Galinier and Hao, 1999], les auteurs proposent d'utiliser le nombre de conflits pour régler dynamiquement la longueur de la liste Tabou, tandis que certains articles plus récents [Blöchliger and Zufferey, 2008; Devarenne *et al.*, 2006] proposent également des techniques réactives.

Dans ce qui suit, nous présentons d'abord les composantes clés d'un algorithme Tabou pour la $k$-coloration, qui servira du point de départ de l'étude présentée dans ce chapitre.

### 2.2.1 Le voisinage

L'espace de recherche $\Omega$ d'un problème de $k$-coloration $(G_{V,E}, k)$ comprend toutes les colorations possibles de $G$ ; comme les configurations sont codées sous forme de vecteurs de couleurs, on obtient $|\Omega| = |V|^k$. Une fonction simple de voisinage $N : \Omega \rightarrow 2^{\Omega} - \{\emptyset\}$ peut être définie comme suit. Étant donnée une $k$-coloration $C$, une $k$-coloration voisine $C'$ peut être obtenue en changeant simplement la couleur $c(i)$ d'un *sommet en conflit* $i$ avec une nouvelle couleur $c'(i)$. La transition de $C$ à $C'$ représente un *mouvement* (un pas) dans un processus de recherche, formellement noté par le couple $< i, c'(i) >$.

Ce voisinage se concentre sur les $|CV|$ sommets en conflit, afin d'aider le processus de recherche à se focaliser sur des mouvements influents, en évitant les mouvements moins pertinents. Habituellement, il n'y a pas besoin de changer la couleur d'un sommet non-conflictuel. La taille de notre voisinage est $|N(C)| = (k-1)|CV|$, une taille considérablement plus petite que la taille d'un voisinage dans lequel *tout* sommet pourrait changer sa couleur (i.e. $(k-1)|V|$).

#### 2.2.1.1 Évaluation rapide du voisinage complet

Pour choisir rapidement la meilleure coloration dans $N(C)$, nous utilisons un tableau $\gamma$ avec $|V|$ lignes et $k$ colonnes : un élément $\gamma_{i,j} = \gamma_{i,c'(i)}$ indique le nombre de conflits que le sommet $i$ aurait si la couleur $j = c'(i)$ lui avait été assignée. La différence du nombre de conflits qui serait induite par un mouvement $< i, c'(i) >$ est $\gamma_{i,c'(i)} - \gamma_{i,c(i)}$. Comme ce voisinage considère seulement des sommets en conflit, le meilleur mouvement est recherché en passant par tous les éléments $\gamma_{i,j}$ avec $i \in CV$ (i.e. $(k-1)|CV|$ éléments de $\gamma$) ; on peut dire que la couleur $c'(i)$ qui sera affectée à $i$ vérifie $c'(i) \in arg\,min_j(\gamma_{i,j} - \gamma_{i,c(i)})$.

Après avoir effectué un mouvement $< i, c'(i) >$, $\gamma$ peut être mis à jour en $O(|V|)$ opérations : il faut modifier *uniquement les colonnes* $c(i)$ *et* $c'(i)$. Cette technique dynamique a été d'abord utilisée dans [Fleurent and Ferland, 1996a] et elle s'avère indispensable pour une investigation rapide du voisinage complet. Un exposé plus général concernant les techniques et les structures de données incrémentales est disponible dans [Galinier, 1999, §6.2]

### 2.2.2 Gestion de la liste Tabou

La liste Tabou est généralement vue comme une structure "first-in-first-out" (une file) qui contient des configurations ou des mouvements récents. Dans notre contexte, il est plus pratique de l'implémenter comme un tableau $T$ de dimension $|V| \times k$ , où chaque élément correspond à un mouvement possible. Chaque fois qu'un mouvement $< i, c'(i) >$ est exécuté, le sommet $i$ reçoit une nouvelle couleur $c'(i)$ et l'ancienne couleur de $i$ – i.e. $c(i)$ – devient interdite (Tabou) pour les prochaines $T_\ell$ itérations (la durée Tabou est $T_l$). Dans la pratique, chaque élément de $T$ indique le nombre courant d'itération *plus* la durée Tabou $T_\ell$. Par conséquent, afin de vérifier si un mouvement $< i, c'(i) >$ est Tabou ou non, $T_{i,c'(i)}$ est comparé avec l'itération courante.

Dans notre version de recherche Tabou, la longueur classique de la liste Tabou est $T_\ell = \alpha \cdot |CE| + \texttt{random}\ (0, A)$, où $\alpha$ et $A$ sont des paramètres calibrés dans des publications précédentes (voir aussi la Section 2.5.1) ; la fonction `random` retourne un entier aléatoire entre 0 et $A$. Dans ce contexte, $\texttt{random}(0, A)$ représente une longueur Tabou générale (fixe par rapport à la qualité), et $\alpha \cdot |CE|$ a le rôle d'interdire plus longtemps les mouvements associés à des configurations de qualité inférieure. Nous introduisons plus tard (Section 2.4) un composant réactif pour ajuster $T_\ell$ en fonction de la fluctuation récente du nombre de conflits.

**Critère d'aspiration** L'algorithme ne peut pas re-effectuer des mouvements Tabou pour une période définie par la longueur $T_\ell$. Dans certains cas, cela peut également empêcher l'exploration de certaines configurations jamais visitées de qualité. Pour contourner ce problème, l'algorithme emploie un simple critère d'aspiration : le statut Tabou d'un mouvement est ignoré si ce mouvement mène à une configuration meilleure que la meilleure jamais visitée.

### 2.2.3 Spécification de l'algorithme Tabou de base

**Algorithme 2.1 :** La recherche Tabou pour $k$-Coloration

**Entrée** : graphe $G$, $k \in I\!N^*$ ;
**Valeur de retour** : $f(C^*)$, le meilleur nombre de conflits jamais trouvé ;
**Variables** :
- $C$ et $C^*$ : la coloration courante et la meilleure coloration trouvée ;
- $T$ et $T_\ell$ : tableau Tabou et durée Tabou ;
- $\gamma$ : tableau incrémental $|V| \times k$ (voir Section 2.2.1.1) ;

**DÉBUT**
1. $I = 0$
2. $T = 0$ /*aucun mouvement n'est Tabou*/
3. $C$ = une $k$-coloration aléatoire
4. $C^* = C$
5. initialiser $\gamma$ /*$\gamma_{i,j}$ = nb. de conflits du $i$ si $i$ serait coloré avec $j$*/
6. **tant que** ($f(C) > 0$ **et** *condition d'arrêt* non atteinte)
   - Sélectionner le meilleur mouvement non-Tabou[a] dans $\gamma$ (Si plusieurs mouvements mènent au même meilleur nombre de conflits, un choix aléatoire est fait utilisant la fonction d'évaluation)
   - $T_{I,c(i)} = I + T_\ell$ /*$I$=l'itération courante*/
   - $c(i) = c'(i)$ (effectuez le mouvement)
   - $f(C) = f(C) + \gamma_{i,c'(i)} - \gamma_{i,c(i)}$
   - Actualiser $\gamma$
   - **si** $(f(C) < f(C^*))$ **alors** $C^* = C$
7. **renvoyer** $f(C^*)$

**FIN**

a. Plus précisément : soit non-Tabou, soit Tabou mais vérifiant le critère d'aspiration.

L'algorithme 2.1 décrit la procédure Tabou de $k$-coloration avec toutes les composantes présentées ci-dessus. Pour une instance de $k$-coloration donnée $(G, k)$, notre algorithme Tabou génère d'abord une première $k$-coloration aléatoire $C$ pour lancer la recherche. Ensuite, les étapes principales d'une itération sont : (i) la sélection du meilleur mouvement acceptable $< i, c'(i) >$ dans le voisinage, (ii) la paire $< i, c(i) >$ devient Tabou, (iii) l'exécution du mouvement $c(i) = c'(i)$ et (iv) la mise à jour de $\gamma$. Le processus s'arrête quand une coloration légale est trouvée ou quand une limite de temps est atteinte.

## 2.3 Nouvelles fonctions d'évaluation "bien informées"

Tous les travaux précédemment cités utilisent directement la fonction $f$ – le nombre de conflits, voir Équation (1.1), p. 17 – comme fonction d'évaluation. Comme $f$ compte seulement le nombre de conflits, elle a un inconvénient inhérent : elle ne fait aucune distinction entre toutes les configurations avec le même nombre de conflits. En effet, ces configurations sont équivalentes pour $f$ même si elles peuvent avoir un potentiel différent pour conduire le processus de recherche vers une amélioration. En termes de paysage de recherche, $f$ peut produire de grands plateaux de colorations complètement équivalentes [Hertz *et al.*, 1994].

Pour surmonter cette difficulté, nous proposons d'enrichir $f$ avec plus d'informations qui peuvent distinguer des configurations équivalentes en termes de nombre de conflits. Nous introduisons une fonction heuristique $h : \Omega \rightarrow [0, 1)$, qui est combinée avec $f$ dans la forme linéaire suivante, menant à une nouvelle fonction d'évaluation $\widetilde{f}$ :

$$\widetilde{f}(C) = f(C) - h(C) \tag{2.1}$$

Considérons deux configurations $C_1$ et $C_2$ telles que $f(C_1) = f(C_2) \neq 0$. Idéalement, nous devrions avoir $\widetilde{f}(C_1) < \widetilde{f}(C_2)$ si la probabilité d'atteindre une solution est plus grande quand la recherche part de $C_1$ que lorsqu'elle part de $C_2$. Malheureusement, en raison de la complexité du paysage de recherche, cette probabilité est inconnue, et le calcul d'une estimation robuste est très difficile. Pour cette raison, il faut se limiter à des informations locales, faciles à calculer et interpréter. Nous proposons deux heuristiques différentes.

La première dépend seulement des informations structurelles du graphe : la distribution des degrés de conflit des *sommets en conflit* (voir ci-dessous). La seconde s'appuie sur un mécanisme d'apprentissage simple utilisant une mémoire à long terme : elle dépend des *fréquences de changement de couleur* sur l'ensemble des sommets pendant une première phase de la recherche.

### 2.3.1 Fonction d'évaluation basée sur le degré

En complément des définitions de la Section 1.2.2, nous présentons de nouvelles définitions utilisées uniquement dans cette section.

**Definition 2.1.** *(Degré de conflit des sommets) Soit $i$ un sommet, $\delta_i$ son degré, et $C$ une coloration. Nous définissons $CV_i = \{j \in V | \{i, j\} \in CE(C)\}$ et nous appelons $\frac{|CV_i|}{\delta_i}$ le degré de conflit du sommet $i$ pour la configuration $C$.*

Il est facile de voir que $0 \leq |CV_i| \leq \delta_i \; \forall i \in V$ ; la valeur minimum $|CV_i| = 0$ est atteinte pour des sommets sans conflit, et la valeur maximum $|CV_i| = \delta_i$ indique que le sommet $i$ est en conflit avec tous ses voisins. D'ailleurs, la relation suivante est valide pour toute coloration : $2|CE| = \sum_{i \in V} |CV_i|$ (voir aussi Définition 1.4, Section 1.2.2).

Pour motiver l'introduction de cette fonction, nous considérons l'exemple de la Figure 2.1 qui montre deux 3-colorations $C_1$ et $C_2$ pour un graphe très simple. Les arêtes en

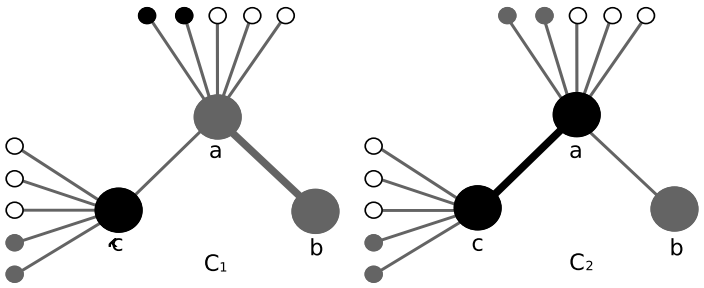


FIGURE 2.1 – Deux 3-colorations $C_1$ et $C_2$ avec un conflit. Le conflit est indiqué avec une ligne plus épaisse dans les deux cas ; il est plus facile de résoudre le conflit gris ($C_1$, à gauche) que de résoudre le noir ($C_2$, à droite) alors que $f(C_1) = f(C_2) = 1$.

conflit sont $\{a, b\}$ pour $C_1$ et, respectivement, $\{a, c\}$ pour $C_2$. Par conséquent, $|CE(C_1)| = |CE(C_2)| = 1$ et les deux configurations sont ainsi équivalentes pour $f$. Cependant, $C_1$ est préférable à $C_2$.

En effet, comme le degré de $b$ est petit, on peut assigner à $b$ une couleur non utilisée par ses voisins (i.e. noir ou blanc) pour résoudre le conflit $\{a, b\}$ sur $C_1$. Cela peut être accompli dans une seule étape sans générer d'autres conflits. Mais il est plus difficile de résoudre le conflit $\{a, c\}$ de $C_2$ car n'importe quel changement de couleur sur le sommet $a$ ou $c$ peut perturber ses nombreux voisins. Intuitivement, plus un sommet a des voisins, plus il est difficile de changer sa couleur sans perturber le reste de la configuration.

Plus généralement, pour chaque sommet en conflit, nous pouvons utiliser son degré pour définir un terme de pénalité : les arêtes en conflits doivent peser plus lourdement dans la fonction d'évaluation si les deux sommets de l'arête ont un degré plus élevé. Afin de prendre en considération tous les sommets en conflit, nous définissons l'heuristique suivante $h_1$ pour associer une pénalité à chaque $k$-coloration $C$ :

$$h_1(C) = \frac{1}{2|E|} \sum_{i \in CV} \frac{|CV_i|}{\delta_i} \tag{2.2}$$

Dans ce contexte, nous voyons que $h_1(C)$ indique la somme des degrés de conflits de tous les sommets de $C$. Notre première fonction d'évaluation basée sur les degrés peut maintenant être définie comme suit :

$$\widetilde{f}_1(C) = f(C) - h_1(C) \tag{2.3}$$

Le seul rôle du coefficient $\frac{1}{2|E|}$ dans l'équation (2.2) ci-dessus est de maintenir la valeur de $h_1$ dans l'intervalle $[0, 1)$. Grâce à cette normalisation, $\widetilde{f}_1$ conserve l'ordre imposé par $f$, i.e. si $f(C) < f(C')$, alors $\widetilde{f}_1(C) < \widetilde{f}_1(C')$. L'objectif de la nouvelle fonction d'évaluation est d'aider le processus de recherche à choisir parmi les voisins avec le même nombre de conflits, et non pas d'introduire des pénalités/poids plus importants que le nombre de conflits.

Certains détails d'implémentation pourraient être utiles ici. À chaque itération, la recherche Tabou de base doit choisir aléatoirement un voisin qui minimise le nombre de conflits. L'implémentation de ce choix aléatoire n'est pas trivial, parce qu'il est assez inefficace de simplement collectionner et enregistrer tous les voisins pour en choisir un finalement. Nous proposons de tirer une valeur aléatoire $r_{C'}$ dans l'intervalle $[0, 1]$ et de l'affecter au voisin $C'$ dès que $C'$ est découvert ; à la fin, le $C'$ avec la plus grande valeur $r_{C'}$ est sélectionné. Cette procédure n'enregistre pas tous les voisins et le choix aléatoire résultant n'est pas biaisé. Vers la fin de la thèse, nous avons réalisé que de meilleurs résultats expérimentaux peuvent être obtenus en couplant les nouvelles fonctions d'évaluation dans ce procédé. Soit $h_{\min}$ et $h_{\max}$ la valeur minimum et maximum atteinte par la fonction $h$. De meilleurs résultats peuvent être réalisés si on tire la valeur $r_{C'}$ dans l'intervalle $\left[0, \frac{h(C')-h_{\min}}{h_{\max}-h_{\min}}\right]$.

### 2.3.2 Des informations dynamiques pour définir d'autres fonctions d'évaluation

Dans cette section, nous proposons une deuxième heuristique $h_2$ qui prend en considération des informations acquises au cours du processus de recherche. L'information considérée est le nombre de changements de couleur par sommet. En principe, si un sommet particulier change sa couleur très fréquemment, le processus de recherche doit se concentrer sur lui en priorité pour lui fixer une couleur appropriée. Plus précisément, nous considérons une première phase de la recherche avec la fonction d'évaluation de base $f$. Pour chaque sommet $i$, nous calculons un *coefficient de fréquence* $freq(i)$ : le nombre de changements de couleur appliqués sur $i$ pendant cette première phase est uniformément normalisé dans un intervalle fixe ("uniform scaling"). L'heuristique $h_2$ et la deuxième nouvelle fonction d'évaluation sont définies comme suit :

$$\widetilde{f}_2(C) = f(C) - h_2(C) = f(C) - \sum_{i \in CV} |CV_i| \cdot \frac{1}{freq(i)} \tag{2.4}$$

En principe, le degré $\delta_i$ (de $\widetilde{f}_1$) est remplacé par un coefficient de fréquence dans cette fonction. En conséquent, étant données deux colorations avec le même nombre de conflits, $\widetilde{f}_2$ préfère celle où les sommets en conflit ont de plus petites fréquences de changement de couleur – cela implique qu'elle préfére résoudre en priorité les sommets de plus haute fréquence de changement. La nouvelle fonction encourage le processus de recherche à résoudre prioritairement les conflits de ces sommets, avant ceux avec de petites fréquences de changement de couleur. En pratique, on peut dire que les sommets avec des hautes fréquences de changement de couleur sont considérés plus critiques. Nous avons observé que $\widetilde{f}_2$ conduit à des changements des couleurs plus fréquents sur les sommets qui étaient "moins modifiés" pendant la première étape ; elle a un effet implicite de diversification.

### 2.3.3 Idées similaires dans la littérature et remarques sur la complexité

Dans la littérature sur la coloration, il y a quelques études utilisant des idées similaires. Par exemple, [Glover *et al.*, 1996] propose de retarder la coloration des "sommets

dépendants" (i.e. de petit degré) dans une dernière étape, après la coloration des autres sommets. L'algorithme Impasse [Morgenstern, 1996] s'appuie sur un codage différent (colorations partielles), mais il prévoit également de colorier d'abord les sommets de degré élevé, laissant les sommets de petit degré dans une classe non coloriée. Cette idée a été également reprise dans [Malaguti *et al.*, 2008].

Concernant la fonction $\widetilde{f}_2$, dans [Devarenne *et al.*, 2006], les sommets "fréquemment" changés pendant N/2 itérations" sont également considérés pour obtenir de la diversité. Des fonctions d'évaluation complètement différentes sont aussi proposées dans la littérature, i.e. [Johnson *et al.*, 1991] ont défini dans leur algorithme de recuit simulé la fonction $\hat{f}_{\text{dsjc}} = -\sum_{i=1}^{k} |C_i| + \sum_{i=1}^{k} 2|C_i||CE_i|$, où $C_i$ est l'ensemble de sommets ayant la couleur $i$ et $CE_i$ est l'ensemble de conflits de la couleur $i$.

#### 2.3.3.1 Complexité de calcul

Un avantage de nos nouvelles fonctions est qu'elles n'introduisent pas de coût de calcul supplémentaire. Notons $h$ l'une des deux heuristiques (voir l'équation 2.1) ; $h$ peut être calculée avec la formule :

$$h(C) = \sum_{i \in V} |CV_i| \cdot h_i = \sum_{\{i,j\} \in CE} (h_i + h_j),$$

où $h_i$ est la pénalité/poids associé au sommet $i$, i.e. $h_i = \frac{1}{2|E| \cdot \delta_i}$ pour la première fonction, ou $h_i = \frac{1}{freq(i)}$ pour la deuxième. En conséquent, comme $f(C) = |CE| = \sum_{\{i,j\} \in CE} 1$, il est possible de calculer la valeur d'une nouvelle fonction $\hat{f}(C)$ en faisant la somme de $1 - (h_i + h_j)$ pour chaque conflit $\{i, j\}$. Avant de lancer la recherche, nous construisons un tableau $E$ tel que $\hat{f} = \sum_{\{i,j\} \in CE} E_{ij}$. La seule différence au niveau du calcul entre les trois fonctions d'évaluation est la valeur initiale de $E$. Ainsi, $E_{ij}$ est toujours égale à 1 pour la fonction classique, mais $E_{ij}$ peut être également définie par $1 - \frac{1}{freq(I)} - \frac{1}{freq(j)}$ pour calculer $\widetilde{f}_2$. Concernant la première fonction, il est important de noter qu'elle peut être également écrite comme :

$$\widetilde{f}_1(C) = \sum_{\{i,j\} \in CE(C)} \left(1 - \frac{1}{2|E|\delta_i} - \frac{1}{2|E|\delta_j}\right) \tag{2.5}$$

## 2.4 Une technique réactive pour gérer la liste Tabou

Il est bien connu que la liste Tabou doit être gérée avec soin. Dans la Section 2.2.2, un mécanisme dynamique a été présenté et nous avons introduit la durée Tabou classique : $T_\ell = \alpha \cdot |CE| + \texttt{random}(0, A)$. Dans cette section, nous renforçons ce mécanisme avec une technique réactive simple mais très efficace contre les problèmes de bouclage.

Considérons les configurations $C_1, C_2, \ldots C_n$ qui font partie d'un *plateau*, i.e. un ensemble de configurations avec la même valeur de fonction objectif, telles que n'importe

quelles deux configurations peuvent être reliées par des mouvements à l'intérieur du plateau. Une liste Tabou de longueur inférieure à $n$ n'est pas suffisante pour couper un cycle de longueur $n : C_1 \rightarrow C_2 \cdots \rightarrow C_n \rightarrow C_1$. Notre longueur classique Tabou est bornée (i.e. $T_\ell$ est toujours inférieure à $\alpha \cdot |CE| + A$) et les tests numériques confirment que cela peut être insuffisant pour éviter certains cycles sur des grands plateaux.

Pour surmonter cette difficulté, nous considérons une liste Tabou réactive : quand le nombre de conflits reste constant pendant un nombre donné d'itérations $M_{\max}$, nous incrémentons la longueur de la liste Tabou pour toutes les itérations suivantes. Autrement dit, dès qu'il y a $M_{\max}$ transitions consécutives $C_1 \rightarrow C_2 \cdots \rightarrow C_{M_{\max}}$ tels que $f(C_1) = f(C_2) = \cdots = f(C_{M_{\max}})$, la liste Tabou devient $T_\ell + 1$ pour les prochaines itérations. Si le nombre de conflits reste toujours constant pendant encore $M_{\max}$ itérations, la liste Tabou devient $T_\ell + 2$ ; ensuite, après encore $M_{\max}$ itérations, elle devient $T_\ell + 3$, etc. La durée Tabou est continuellement incrémentée tant qu'il n'y a pas de variation dans le nombre de conflits. Comme la liste est remise à la valeur originale dès que le nombre de conflits change, nous garantissons que, tôt ou tard, la liste croît jusqu'à une valeur qui peut couper un cycle de n'importe quelle longueur $n$.

Notons que l'utilisation d'une longue liste Tabou pendant tout le processus de recherche aurait eu un effet très négatif ; en dehors des grands plateaux, cela pourrait encourager l'algorithme à abandonner trop tôt des régions prometteuses. Dans notre cas, l'algorithme apprend de sa propre évolution, et il utilise une liste Tabou plus grande seulement quand cela est nécessaire. Ce mécanisme simple a permis à l'algorithme de se débloquer dans des situations critiques sans affecter la recherche en dehors des plateaux. En conclusion, notre durée Tabou peut être exprimée par la formule :

$$T_\ell = \alpha |CE| + \texttt{random}(0, A) + \frac{M_{\text{cst}}}{M_{\max}}, \tag{2.6}$$

où $M_{\text{cst}}$ est le nombre des dernières itérations avec un nombre de conflits *constant*.

Pour rapporter la contribution actuelle à des travaux de recherche antérieurs, cette stratégie Tabou s'inscrit dans le cadre d'algorithmes "Reactive Tabou Search" [Battiti *et al.*, 2008]. D'autres stratégies réactives basées sur la détection de bouclage ont été indépendamment étudiées dans d'autres algorithmes récents de coloration [Blöchliger and Zufferey, 2008; Devarenne *et al.*, 2006]. L'idée la plus similaire a certainement été présentée dans [Blöchliger and Zufferey, 2008] : le processus de recherche est considéré "bloqué" si la fluctuation de la fonction objectif reste en dessous d'un certain seuil pour une longue période. Cette méthode nécessite trois paramètres qui sont réglés dans une phase spéciale de calibrage. Les auteurs de [Devarenne *et al.*, 2006] considèrent des idées différentes et ils se concentrent sur l'identification des sommets causant des boucles. De détails supplémentaires sur l'influence pratique de notre réglage réactif sont donnés dans la partie expérimentale (Section 2.5.3).

## 2.5 Résultats et discussions

Dans cette section, nous présentons des résultats empiriques de la Recherche Tabou Renforcée de Coloration (RCTS) sur l'ensemble complet des graphes DIMACS. Le but

principal est d'évaluer l'influence des nouvelles fonctions d'évaluation, mais aussi l'effet de la liste Tabou réactive.

L'ensemble complet des graphes DIMACS a été présenté dans la Section 1.2.3 (p. 17). Nous rappelons que les algorithmes de coloration traitent des instances $(G, k)$, où $k$ est donné pour chaque graphe. Le niveau de difficulté pour trouver une $k$-coloration légale peut varier de trivial à très difficile. D'abord, notons que RCTS a pu résoudre assez facilement (i.e. avec un taux de réussite de 100% en quelques minutes au maximum) toutes les *instances faciles* de la section 1.2.3.1, ainsi que l'instance ($r1000.1c$, $k = 98$). Dans ce qui suit, nous nous concentrons seulement sur le reste des instances, celles qui sont réellement difficiles pour RCTS.

### 2.5.1 Paramètres

Rappelons que l'algorithme RCTS nécessite seulement trois paramètres : $A, \alpha$ et $M_{\max}$. Les deux premiers sont hérités des algorithmes Tabou précédents, et le dernier est employé par la nouvelle liste Tabou réactive (Section 2.4) :

- $A$ et $\alpha$ : ils sont utilisés pour calculer la longueur de la liste Tabou ($T_\ell = \alpha \cdot |CE| + \texttt{random}(0, A)$, voir la Section 2.2.2). Nous affectons à ces paramètres des valeurs déjà présentées dans la littérature [Galinier and Hao, 1999; Dorne and Hao, 1998b] : $A = 10$ et $\alpha = 0,6$. Nos propres tests expérimentaux de calibrage confirment que cette combinaison constitue un bon choix ;
- $M_{\max}$ : après $M_{\max}$ itérations avec nombre de conflits $|CE|$ constant, le processus de recherche est considéré coincé, et ainsi, la composante réactive est activée (cf. Section 2.4). Dans nos tests expérimentaux, $M_{\max}$ est toujours fixé à 1000 pour toutes les instances, mais ce choix est robuste : il existe de nombreuses autres valeurs $M_{\max}$ qui conduisent aux mêmes résultats. Nous avons empiriquement observé que, si $|CE|$ reste constant pour 1000 itérations, $|CE|$ reste constant indéfiniment (avec la liste Tabou classique). Si une valeur plus grande $M_{\max}$ est utilisée, la seule différence est que la réaction est moins prompte. Des valeurs $M_{\max}$ plus petites peuvent déclencher des réactions plus rapidement que nécessaire, influençant alors le processus de recherche sans raison.

### 2.5.2 Influence des fonctions d'évaluation

Dans cette section, nous présentons des résultats expérimentaux pour évaluer l'utilité des nouvelles fonctions d'évaluation.

#### 2.5.2.1 Influence sur l'algorithme Tabou

Une démarche simple pour suivre et comprendre l'évolution d'un algorithme consiste à analyser son profil d'exécution. Nous montrons que les nouvelles fonctions d'évaluation permettent à la recherche de visiter (en moyenne) des configurations de meilleure qualité.

La Figure 2.2 montre une *évolution typique* (sur un graphe aléatoire) du nombre de conflits avec $f$ *et* $\widetilde{f}_1$ pour les 25,000 premières itérations. Le profil de $\widetilde{f}_2$-RCTS n'est pas montré, car il est habituellement superposé au graphique de $f$-RCTS. Ce test expérimental

prouve que $\widetilde{f}_1-$ RCTS reste le plus souvent à un nombre inférieur de conflits. Ce type de profil est observé sur plusieurs graphes, fournissant un argument préliminaire que $\widetilde{f}_1$ peut mener la recherche vers des configurations de meilleure qualité.

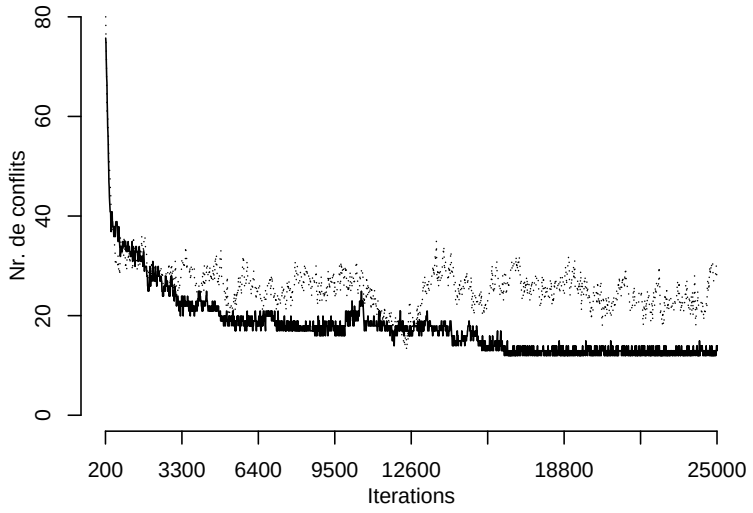


FIGURE 2.2 – Évolution typique du nombre de conflits (profile d'exécution) avec $f$ (en ligne pointillée) et $\widetilde{f}_1$ (en ligne continue) de l'itération 200 jusqu'à l'itération 25.000. $\widetilde{f}_1$ guide la recherche vers de meilleures colorations.

Le Tableau 2.1 confirme cette remarque : il montre le nombre moyen de conflits après 1.000.000 d'itérations avec chaque fonction sur plusieurs graphes représentatifs de différentes familles. Les différences entre ces moyennes ont été confirmées par un test statistique. Nous avons considéré l'hypothèse nulle selon laquelle la moyenne du nombre de conflits obtenus avec $\widetilde{f}_1$ (ou $\widetilde{f}_2$ respectivement) est égale à la moyenne obtenue avec $f$. Utilisant un niveau de confiance de $\alpha = 0.1\%$, cette hypothèse a été rejetée dans la plupart des cas, confirmant que les différences de moyenne ne sont pas dues au hasard – voir les deux dernières colonnes du Tableau 2.1.

Le Tableau 2.1 montre que le nombre moyen de conflits est statistiquement plus petit pour $\widetilde{f}_1$ que pour $\widetilde{f}_c$ ; en fait, la différence est statistiquement significative pour tous les graphes sauf pour $flat300\_28$. Dans le meilleur des cas, la moyenne du nombre de conflits pour $\widetilde{f}_1$-RCTS peut être égale à la moitié de la moyenne pour $f$-RCTS. La deuxième fonction $\widetilde{f}_2$ montre également une amélioration sur deux-tiers des instances, mais avec une plus petite amplitude.

Pour le problème de coloration, il n'y a aucune garantie théorique que la probabilité d'atteindre une solution optimale soit strictement corrélée avec la qualité moyenne des solutions visitées tout le long de la recherche. Cependant, particulièrement pour les recherches locales, presque tous les algorithmes essayent toujours de guider le processus vers des configurations de plus grande qualité (i.e. avec moins de conflits).

| Graphe | $k$ | Nombre moyen de conflits | | | Test statistique | |
|---|---|---|---|---|---|---|
| | | $\widetilde{f}_1$ | $\widetilde{f}_2$ | $f$ | $[\widetilde{f}_1] \not\equiv [f]$ | $[\widetilde{f}_2] \not\equiv [f]$ |
| $dsjc250.5$ | 28 | 8.138 | 7.133 | 9.878 | Oui | Oui |
| $dsjc500.5$ | 48 | 24.9 | 28.5 | 26.6 | Oui | Oui |
| $dsjc1000.5$ | 87 | 35 | 32.2 | 37.7 | Oui | Oui |
| $dsjr500.5$ | 122 | 5.106 | 8.875 | 10.46 | Oui | Oui |
| $r1000.5$ | 234 | 16.02 | 25.76 | 29.09 | Oui | Oui |
| $flat300\_280$ | 30 | 21.63 | 23.87 | 22.13 | Non | Non |
| $flat1000\_76$ | 86 | 30.39 | 29.02 | 32.04 | Oui | Oui |
| $le450\_25c$ | 25 | 9.38 | 9.524 | 12.97 | Oui | Oui |
| $le450\_25d$ | 25 | 9.183 | 13.94 | 13.74 | Oui | Non |

TABLE 2.1 – Moyenne du nombre de conflits après 1.000.000 itérations avec chaque fonction. En général, les fonctions modifiées, en particulier $\widetilde{f}_1$, mènent la recherche vers des colorations avec (statistiquement) moins de conflits.

#### 2.5.2.2 Fonction d'évaluation avec la liste Tabou vide

Nous sommes convaincus que l'intérêt d'une fonction d'évaluation plus discriminante n'est pas limitée à l'algorithme Tabou ou à un certain réglage de la liste Tabou. Nous avons donc effectué un autre test expérimental avec un algorithme de Descente Pure (DP) sans paramètres, plus neutre. Techniquement, l'algorithme DP est la même recherche Tabou de la Section 2.2, mais avec une liste Tabou toujours vide (i.e. $T_\ell = 0$). Cette descente pure est initialisée par une coloration aléatoire et elle est guidée seulement par la fonction d'évaluation : DP choisit itérativement le meilleur voisin selon cette fonction. Un nombre très petit d'itérations (i.e. moins que quelques milliers) est généralement suffisant pour faire converger la recherche dans des optima locaux. Comme une étape d'apprentissage de quelques milliers d'itérations serait insuffisante pour $\widetilde{f}_2$, ce test ne peut être effectué qu'avec $\widetilde{f}_1$ et $f$.

Nous avons lancé 1000 descentes indépendantes (chacune avec une coloration initiale différente) et nous avons comparé le nombre de conflits des $k$-colorations finalement obtenues avec $\widetilde{f}_1$ et $f$. La Figure 2.3 montre les distributions de ces nombres de conflit (i.e. l'axe $x$ indique le nombre de conflits et l'axe $y$ indique la fréquence de ce nombre – le nombre de descentes qui sont arrivées à ce nombre) : la performance avec $\widetilde{f}_1$ est nettement supérieure et même la plus mauvaise descente avec $\widetilde{f}_1$ est souvent meilleure que la meilleure avec $f$. Nous avons observé ces distributions pour plusieurs graphes de toutes les familles. La nouvelle fonction permet à cette descente rudimentaire d'atteindre une solution pour $G = le450\_25a$ et $k = 25$, tandis que la même descente avec $f$ ne trouve jamais de coloration avec moins de 10 conflits.

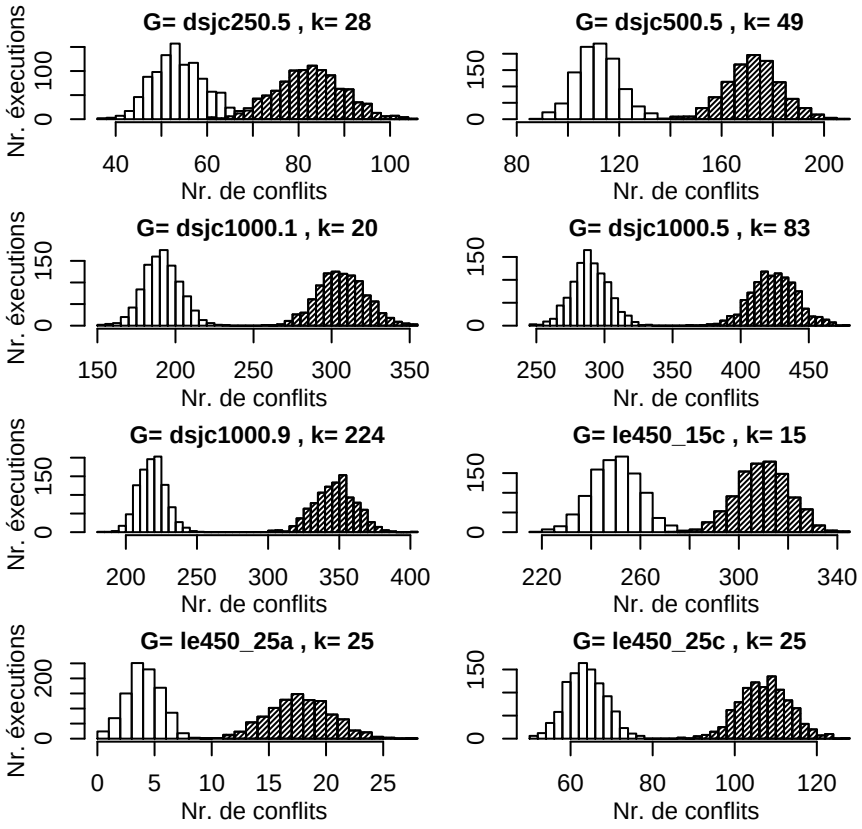


FIGURE 2.3 – Histogrammes du nombre de conflits obtenus avec 1000 descentes pures indépendantes, utilisant $\widetilde{f}_1$ (barres simples) et $f$ (barres hachurées). La descente pure mène toujours à un plus petit nombre de conflits avec $\widetilde{f}_1$ que avec $f$.

#### 2.5.2.3 Variation de performance selon les caractéristiques des instances

Nous avons également observé que l'influence favorable de la nouvelle fonction $\widetilde{f}_1$ est plus évidente sur certaines instances que sur d'autres. En principe, les meilleures améliorations sont réalisées sur des instances difficiles de certaines classes spécifiques. Une amélioration impressionnante est observée sur les graphes géométriques aléatoires (i.e. $dsjrX.Y$ et $rX.Y$ ) pour lesquels $\widetilde{f}_1$ domine fortement $f$ sans exception – voir le Tableau 2.1 et également le Tableau 2.2 avec des résultats complets.

Cette variation de performance est due à la structure des graphes, plus exactement à la variation de degrés de sommets. Par exemple, pour les graphes géométriques (e.g. $dsjr500.5$), le degré maximum peut être d'un ordre de grandeur plus grand que le degré minimum, et ainsi toute différentiation fondée sur le degré est très efficace. En effet, l'amélioration des performances apportée par $\widetilde{f}_1$ peut être classée en concordance avec la variation des degrés, du plus haut au plus bas : 1) graphes géométriques aléatoires, 2) graphes de Leighton, 3) graphes aléatoires, 4) graphes flat. Un cas extrême est le graphe

du carré latin *latin_square* qui est régulier (i.e. avec degré constant) ; la nouvelle fonction d'évaluation basée sur le degré n'apporte pas de nouvelle distinction entre les sommets de ce graphe.

### 2.5.3 Influence de la liste Tabou réactive

Pour évaluer l'influence du réglage réactif, nous avons analysé et comparé la recherche Tabou classique (Section 2.2.2) avec la recherche à durée Tabou réactive. Comme nous l'avons précisé précédemment, l'objectif de la composante réactive est d'éviter les problèmes de bouclage sur des colorations plateaux. Utilisant plusieurs graphes représentatifs de chaque famille importante, nous avons lancé entre 20 et 100 recherches Tabou sans réglage réactif et nous avons compté combien d'exécutions se sont coincées avant d'arriver à 20 millions d'itérations. Nous considérons que le processus de recherche est coincé s'il arrive à un moment à partir duquel le nombre de conflits reste constant indéfiniment (jusqu'à la fin des 20 millions d'itérations allouées). La conclusion de ce test a été très claire : plus de 90% des exécutions se sont trouvées coincées dans une boucle sur un plateau avant d'atteindre les 20 millions d'itérations.

Avec le réglage réactif, ces problèmes de bouclage sont résolus et l'algorithme Tabou peut échapper avec succès à la plupart des plateaux. Ainsi, notre algorithme peut effectivement bénéficier d'un temps d'exécution plus élevé – sans réglage réactif, il serait en fait inutile de faire tourner la recherche au-delà d'un certain nombre d'itérations (pour toute fonction d'évaluation). Un point positif de notre réglage réactif est la simplicité : un seul paramètre $M_{\max}$ est nécessaire et on peut lui affecter une valeur pertinente avec un minimum d'effort expérimental. Ce paramètre doit satisfaire uniquement la condition suivante : si $|CE|$ reste constant pour $M_{\max}$ itérations, alors $|CE|$ reste constant pour une période illimitée. Nous avons empiriquement observé que la composante réactive est déclenchée seulement quelques fois sur des millions d'itérations (sur des plateaux qui autrement bloqueraient le processus) ; ainsi, l'interférence avec d'autres composantes de l'algorithme est minimale.

### 2.5.4 Résultats complets sur toutes les instances difficiles

Dans cette section, nous présentons des résultats détaillés de RCTS avec les trois fonctions d'évaluation pour toutes les instances DIMACS difficiles. Pour chaque paire $(G, k)$, nous avons lancé 10 exécutions indépendantes et nous rapportons le taux de réussite, ainsi que l'effort moyen de calcul (mesuré en itérations et en temps CPU) pour trouver une coloration légale. La condition d'arrêt est d'atteindre 10 heures sur un processeur Xeon à 2.8GHz utilisant le langage de programmation C++ compilé avec l'option d'optimisation -O2 (version 4.1.2 de GCC sous Linux).

Nous avons observé que, dans la littérature sur la coloration, il est une pratique courante de faire tourner un algorithme au minimum quelques heures. Par exemple, un temps limite de 10 heures est également utilisé dans [Hertz *et al.*, 2008, Table 5] et [Blöchliger and Zufferey, 2008, Table 6] – et avec un nombre d'itérations allant jusqu'à 2000 millions. De plus, un autre algorithme récent très performant emploie un temps de plus

de 10 heures (e.g. 40000 secondes dans [Malaguti *et al.*, 2008, p.310]). Nous trouvons également des articles rapportant plusieurs jours de calcul (e.g. [Dorne and Hao, 1998a; Morgenstern, 1996]).

| Graph | k | $\widetilde{f}_1$-RCTS | | | $f$-RCTS | | | $\widetilde{f}_2$-RCTS | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | #Hits [/ 10] | #Iters [$10^6$] | Time [h] | #Hits [/ 10] | #Iters [$10^6$] | Time [h] | #Hits [/ 10] | #Iters [$10^6$] | Time [h] |
| $dsjc500.1$ | 12 | 10/10 | 96 | $< \frac{1}{4}$ | 10/10 | 64 | $< \frac{1}{4}$ | 10/10 | 64 | $< \frac{1}{4}$ |
| $dsjc500.5$ | 48 | 1/10 | 1352 | 6.33 | 0/10 | — | — | 0/10 | — | — |
| $dsjc500.9$ | 126 | 9/10 | 360 | 1.67 | 10/10 | 457 | 2.4 | 10/10 | 466 | 2.5 |
| $dsjc1000.1$ | 21 | 10/10 | 1 | $< \frac{1}{4}$ | 10/10 | 2 | $< \frac{1}{4}$ | 10/10 | 2 | $< \frac{1}{4}$ |
| $dsjc1000.5$ | 87 | 1/10 | 873 | 7 | 0/10 | — | — | 0/10 | — | — |
| $dsjc1000.9$ | 224 | 4/10 | 420 | 7.5 | 1/10 | 321 | 5 | 3/10 | 492 | 7.33 |
| $dsjr500.1c$ | 85 | 5/10 | 470 | 1.2 | 1/10 | 7 | $< \frac{1}{4}$ | 1/10 | 7 | $< \frac{1}{4}$ |
| $dsjr500.5$ | 122 | 7/10 | 469 | 2.4 | 0/10 | — | — | 0/10 | — | — |
| $r250.5$ | 65 | 10/10 | 99 | $< \frac{1}{4}$ | 0/10 | — | — | 0/10 | — | — |
| $r1000.5$ | 237 | 2/10 | 1059 | 7.5 | 0/10 | — | — | 0/10 | — | — |
| $flat1000.76$ | 87 | 10/10 | 290 | 2.3 | 10/10 | 265 | 2 | 10/10 | 265 | 2 |
| $flat300.28$ | 30 | 4/10 | 1183 | 4.5 | 6/10 | 538 | 2.33 | 8/10 | 737 | 3.1 |
| $le450.15c$ | 16 | 8/10 | 11 | $< \frac{1}{4}$ | 9/10 | 17 | $< \frac{1}{4}$ | 9/10 | 17 | $< \frac{1}{4}$ |
| $le450.15d$ | 16 | 10/10 | 1 | $< \frac{1}{4}$ | 10/10 | $< 1$ | $< \frac{1}{4}$ | 10/10 | $< 1$ | $< \frac{1}{4}$ |
| $le450.25c$ | 25 | 9/10 | 621 | 1.1 | 6/10 | 572 | 1.2 | 6/10 | 203 | 1.2 |
| $le450.25d$ | 25 | 9/10 | 937 | 2 | 2/10 | 1895 | 4.5 | 2/10 | 1895 | 4.5 |
| $latin\_square$ | 100 | 6/10 | 641 | 3 | 4/10 | 1005 | 5 | 5/10 | 1141 | 5.5 |
| $C2000.5$ | 162 | 4/10 | 237 | 3.63 | 1/10 | 601 | 9.5 | 2/10 | 477 | 8 |
| $C4000.5$ | 305 | 4/10 | 88 | 4.75 | 2/10 | 85 | 4.5 | 2/10 | 98 | 5.5 |

TABLE 2.2 – Résultats détaillés de RCTS avec un temps limite de 10 heures avec les trois fonctions d'évaluation. $\widetilde{f}_1$ - RCTS trouve de meilleures solutions ($k$ plus petit) que $f$ - RCTS sur 25% de graphes et il obtient un taux de succès amélioré sur 25% des autres instances. La différence entre $\widetilde{f}_2$ - RCTS et $f$ - RCTS est plus nette sur les grands graphes, qui nécessitent plus de 5h de calcul.

Le Tableau 2.2 présente les résultats détaillés de RCTS sur tous les graphes DIMACS difficiles. Les deux premières colonnes indiquent l'instance de $k$-coloration, i.e. le graphe et le nombre de couleurs $k$. Pour chaque fonction d'évaluation, nous fournissons le taux de réussite (colonnes 3,6,9) et l'effort moyen de calcul nécessaire pour résoudre l'instance : le nombre moyen d'itérations en millions (colonnes 4, 7, et 10) et le temps de calcul en heures (colonnes 5, 8, 11). Le nombre d'itérations est une mesure indépendante de la machine et le temps est indiqué pour avoir une idée de l'ordre de grandeur ; le symbole "-" indique

qu'aucune solution n'a été trouvée en 10 heures.

Tout d'abord, nous observons sur le Tableau 2.2 que $\widetilde{f}_1$-RCTS trouve des meilleurs $k$-colorations (i.e. avec un plus petit $k$) que $f$-RCTS pour plus de 25% des instances – i.e. 5 graphes sur 19. Comme nous l'avons déjà souligné en Section 1.2.3.2, cette différence d'une couleur est régulièrement associée à une diminution de *plusieurs* conflits. De plus, pour certains graphes, nous nous demandons s'il est effectivement possible de réduire le nombre de couleurs avec d'autres méthodes (voir aussi les meilleurs algorithmes dans le Tableau 2.3). Deuxièmement, $\widetilde{f}_1$ - RCTS obtient un taux de succès deux fois meilleur que $f$-RCTS pour 25% des autres instances. Ainsi, on peut dire que la nouvelle fonction d'évaluation $\widetilde{f}_1$ apporte une amélioration importante pour 10 instances sur un total de 19. Nous pouvons également observer que la meilleure amélioration est survenue pour les instances les plus difficiles et pour les classes de graphes ayant une forte variation de degré (voir aussi Section 2.5.2.3).

En effet, la meilleure performance est réalisée sur les graphes géométriques aléatoires pour lesquels $\widetilde{f}_1$-RCTS peut trouver rapidement (en moins d'une heure) des solutions que $f$-RCTS ne trouve pas en 10 heures – e.g. voir le graphe $r250.5$. D'ailleurs, $\widetilde{f}_1$-RCTS résout également l'instance difficile ($dsjr500.5$, $k = 122$) en moins de deux heures avec un taux de réussite stable. Parmi les dix meilleurs algorithmes de la littérature (voir également le Tableau 2.3), cette instance était précédemment résolue uniquement par deux algorithmes beaucoup plus complexes [Malaguti *et al.*, 2008; Lü and Hao, 2010]. Comme nous l'avons précisé dans la Section 2.5.2.3, la puissance de discrimination de $\widetilde{f}_1$ est moins visible pour les graphes ayant une variation de degré très faible (e.g. graphes $flatX.Y$).

L'avantage de $\widetilde{f}_2$ sur $f$ se voit sur les plus grands graphes ($dsjc1000.9$, $latin\_square$, $C2000.5$, $C4000.5$). Pour ces graphes, les meilleurs résultats sont généralement obtenus après 5 heures de calcul (c'est à dire, avec la fonction $\widetilde{f}_2$ pendant la deuxième partie du temps total de 10 heures) et la fonction d'évaluation $\widetilde{f}_2$ améliore le taux de succès dans 75% des cas. Pour les autres graphes, il n'y a pas une différence majeure de performance, simplement parce que la meilleure $k$-coloration est régulièrement trouvée en moins de 5h.

## 2.6 Conclusions

Nous avons présenté une Recherche Tabou Renforcée de Coloration (RCTS) qui améliore les algorithmes Tabou précédents avec deux nouvelles fonctions d'évaluation et avec une liste Tabou réactive. En enrichissant la fonction d'évaluation conventionnelle $f$, les nouvelles fonctions prennent en considération des informations supplémentaires relatives à la structure du graphe (degrés de conflit des sommets) ainsi que des informations dynamiques acquises au cours de la recherche (la fréquence de changement de couleur). De plus, la gestion réactive de la liste Tabou permet à l'algorithme d'éviter le bouclage ; ainsi, il peut effectivement profiter d'un temps de calcul plus long. Nous avons montré que l'algorithme résultant, en dépit de sa simplicité, est suffisant pour atteindre, à de nombreuses reprises, les meilleurs résultats de la littérature.

Le Tableau 2.3 compare les résultats de RCTS avec les meilleurs algorithmes de la littérature – cinq d'entre eux sont basés sur une recherche locale et six sont des algorithmes

| Graphe | $\chi/k^*$ | | Recherches Locales [a] | | | | | Algorithmes Hybrides [a] | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | RCTS | ILS 2002 | VNS 2003 | ALS 2008 | PCol 2008 | VSS 2008 | DCNS 1996 | HGA 1996 | HEA 1999 | AMCol 2008 | MMT 2008 | MCol 2010 |
| $dsjc500.1$ | ?/12 | 12 | 12 | – | 13 | 12 | 12 | – | – | – | 12 | 12 | 12 |
| $dsjc500.5$ | ?/48 | 48 | 49 | 49 | 50 | 48 | 48 | 49 | 49 | 48 | 48 | 48 | 48 |
| $dsjc500.9$ | ?/126 | 126 | 126 | – | 128 | 126 | 126 | – | – | – | 126 | 127 | 126 |
| $dsjc1000.1$ | ?/20 | 21 | – | – | 21 | 20 | 20 | – | – | 20 | 20 | 20 | 20 |
| $dsjc1000.5$ | ?/83 | 87 | 89 | 90 | 89 | 89 | 88 | 89 | 84 | 83 | 84 | 83 | 83 |
| $dsjc1000.9$ | ?/224 [223] | 224 | – | – | 230 | 225 | 224 | 226 | – | 224 | 224 | 225 | 223 |
| $dsjr500.1c$ | 84/85 | 85 | – | – | – | 85 | 85 | 85 | 85 | – | 86 | 85 | 85 |
| $dsjr500.5$ | 122/122 | 122 | 124 | – | – | 126 | 125 | 123 | 130 | – | 125 | 122 | 122 |
| $r250.5$ | 65/65 | 65 | – | – | – | 66 | – | 65 | 69 | – | – | 65 | 65 |
| $r1000.5$ | 234/234 | 237 | – | – | – | 248 | – | 241 | 268 | – | – | 234 | 245 |
| $flat300.28$ | 28/28 | 30 | 31 | 31 | – | 28 | 28 | 31 | 33 | 31 | 31 | 31 | 29 |
| $flat1000.76$ | 76/82 | 87 | – | 89 | – | 88 | 86 | 89 | 84 | 83 | 84 | 82 | 82 |
| $le450.15c$ | 15/15 | 16 | 15 | 15 | – | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 |
| $le450.15d$ | 15/15 | 16 | 15 | 15 | – | 15 | 15 | 15 | 15 | – | 15 | 15 | 15 |
| $le450.25c$ | 25/25 | 25 | 26 | – | – | 25 | 26 | 25 | 25 | 26 | 26 | 25 | 25 |
| $le450.25d$ | 25/25 | 25 | 26 | – | – | 25 | 26 | 25 | 25 | – | 26 | 25 | 25 |
| $latin\_square$ | ?/98 | 100 | 99 | – | – | – | – | 98 | 106 | – | 104 | 101 | 99 |
| $C2000.5$ | ?/150 [148] | 162 | – | – | – | – | – | 150 | 153 | – | – | – | 148 |
| $C4000.5$ | ?/280 [272] | 305 | – | – | – | – | – | – | 280 | – | – | – | 272 |

TABLE 2.3 – Résultats du RCTS et des meilleurs algorithmes les plus performants dans la littérature sur tous les graphes DIMACS difficiles. Les colorations trouvées par RCTS sont publiquement disponibles à l'adresse : `www.info.univ-angers.fr/pub/porumbel/graphs/rcts/`.

*a.* Les acronymes des algorithmes se trouvent dans la Section 1.2.4 (p. 19). Comme précisé dans cette section, on indique entre crochets les bornes publiés au cours de la rédaction de ce manuscrit.

hybrides. Même si une comparaison détaillée est au-delà des objectifs de ce chapitre, ce tableau de comparaison donne une vision générale indiquant une bonne performance de RCTS sur l'ensemble des instances DIMACS.

**Fonction d'évaluations dans un contexte général** Finalement, il est important de noter que les fonctions d'évaluation proposées peuvent être directement utilisées par d'autres algorithmes de coloration. Plus généralement, il semble que l'idée d'utiliser une fonction d'évaluation spécifique au problème a été, dans une certaine mesure, négligée jusqu'à récemment. Nous sommes convaincus qu'une fonction d'évaluation soigneusement conçue – utilisant des connaissances spécifiques du problème – peut améliorer la résolution heuristique des autres problèmes d'optimisation combinatoire, comme cela a déjà été démontré pour certains cas [Rodriguez-Tello *et al.*, 2008a; Rodriguez-Tello *et al.*, 2008b].

# Chapitre 3

# Cartographie de l'Espace de Recherche

Ce chapitre est consacré à l'analyse de l'espace de recherche : nous nous focalisons sur la distribution spatiale des configurations de qualité. Les informations apprises dans cette étape sont intégrées dans nos heuristiques afin de leur fournir d'avantage d'information : les recherches locales du Chapitre 4 peuvent s'auto-guider vers des régions non-explorées, l'algorithme évolutionniste du Chapitre 5 utilise une population avec un écart approprié entre les individus. En utilisant une mesure de distance entre les colorations, nous présentons l'hypothèse de *clusterisation* des solutions candidates de l'espace de recherche : les configurations de grande qualité ne sont *pas* distribuées aléatoirement, mais elles sont plutôt groupées en clusters, dans des sphères de diamètre spécifique. Tout d'abord, nous fournissons des arguments très intuitifs en projetant un échantillon d'optima locaux dans un espace 3D. Puis, nous étudions formellement la distribution de la distance entre les meilleures configurations visitées par une recherche locale pendant une longue période : les valeurs de distance sont soit très petites (distances à l'intérieur d'un clusters), soit très grandes (distances entre les cluster). Ce chapitre développe des idées d'un article accepté par *Computers and Opperations Research* [Porumbel *et al.*, 2010] ; une partie de l'état de l'art sur l'analyse d'espace provient d'un article présenté à la conférence *LION (Learning In OptimizatioN)* [Porumbel *et al.*, 2009d].

## Sommaire


**3.1 Introduction . . . . . 42**
3.1.1 Analyse de l'espace de recherche . . . . . 42
**3.2 La recherche locale typique et la vision globale . . . . . 43**
**3.3 Cartographie de l'espace de recherche . . . . . 45**
3.3.1 L'opération de cartographie . . . . . 45

3.3.2 Distribution spatiale des meilleurs optima . . . . . 47
3.3.3 Distribution spatiale des configurations visitées en série sur une courte période . . . . . 48
3.3.4 Distribution spatiale des colorations visitées en série sur une longue période . . . . . 50
**3.4 Conclusion du chapitre . . . . . 51**

## 3.1 Introduction

### 3.1.1 Analyse de l'espace de recherche

Il est bien connu que l'évolution de tout algorithme heuristique est fortement influencée par la structure de l'espace de recherche. En fait, pour concevoir une heuristique efficace, il est essentiel d'exploiter (implicitement ou explicitement) les propriétés de l'espace de recherche [Streeter and Smith, 2006]. Par exemple, plusieurs structures différentes d'optima locaux peuvent être exemplifiées pour le problème de satisfaisabilité en logique propositionnelle (SAT) [Du and Pardalos, 2007, pp. 425–427] : optima locaux isolés, plateaux, structures de vallées, bassins d'attraction, etc. Pour de nombreux algorithmes, la situation la plus difficile n'est pas le minimum local classique, mais le "piège" (trap, en anglais) : le groupement de plusieurs optima locaux dans un "puits" profond – on peut voir un "puits" comme un trou dans la surface déterminée par la fonction objectif ("fitness surface" en anglais). Une fois piégée dans une telle structure, une recherche locale classique va boucler en permanence entre les minima locaux à l'intérieur du puits bien qu'elle puisse avoir les capacités d'échapper à tout optimum local individuel.

Il existe plusieurs pistes de recherche concernant l'analyse des espaces de solutions – voir aussi la classification de [Streeter and Smith, 2006, §2] qui est bien reliée à nos intérêts. Pour illustration, dans le contexte de calcul évolutionniste, une direction de recherche active est centrée sur des indicateurs statistiques pour estimer la difficulté du problème – i.e. convexité, rugosité, "smoothness", "fitness-distance correlation" [Jones and Forrest, 1995] ; on peut se référer à [Kallel *et al.*, 2001; Merz, 2004] pour un résumé de tels indicateurs et de discussions connexes. D'autres types d'analyses d'espace de recherche traitent des similitudes structurales entre les optima locaux (i.e. "backbone structures"), ou leur distribution spatiale – comme nous le faisons dans ce chapitre.

Chaque problème d'optimisation peut avoir un nombre différent d'optima locaux et de plateaux – chacun d'entre eux avec sa propre forme, taille, profondeur, etc. De plus, ces optima locaux peuvent être groupés d'après des modèles spécifiques (bassins d'attraction, vallées, puits) et ils peuvent être concentrés dans quelques régions [Merz, 2004]. On peut trouver de nombreuses études relatives à des propriétés spécifiques pour plusieurs problèmes représentatifs : le problème SAT [Zhang, 2004; Gerber *et al.*, 1998], le voyageur de commerce [Stadler and Schnabl, 1992], le sac-à-dos en 0–1 [Ryan, 1995], la coloration de graphe [Hertz *et al.*, 1994; Hamiez and Hao, 2004; Culberson and Gent, 2001], le bi-partitionnement de graphe [Merz and Freisleben, 2000b], l'affectation quadratique [Merz and Freisleben, 2000a], l'ordonnancement de tâches

[Reeves and Yamada, 1998; Streeter and Smith, 2006], la minimisation de croisements pour la trace de graphes [Kuntz *et al.*, 2004]. Les caractéristiques de l'espace de recherche (le nombre d'optima, leur distribution, la topologie de leurs bassins d'attraction, etc.) se sont révélées en effet assez différentes d'un problème à l'autre, et même d'une instance à l'autre pour le même problème. Néanmoins, toutes ces études concluent que l'analyse de l'espace peut apporter un impact très positif sur le comportement des algorithmes.

Comme nous l'avons discuté en Section 1.1.2, il est plus difficile de faire une analyse complexe "on the fly" (apprentissage en ligne) que de la faire en pré-optimisation, avant l'étape principale de recherche. L'inconvénient de toute analyse d'optima locaux en pré-optimisation est évident : cela exige de connaître les optima locaux en préalable – la localisation des optima locaux de grande qualité est en fait l'objectif final de la recherche principale. Avant l'étape principale d'optimisation, très peu d'informations sont habituellement disponibles sur l'espace de recherche du problème. Une approche possible serait d'utiliser de petites instances afin de localiser facilement tous les optima locaux pour les analyser ensuite [Hertz *et al.*, 1994]. Une autre approche très populaire consiste à analyser le comportement des heuristiques sur des paysages de recherche artificiels, e.g. le modèle NK [Kauffman and Levin, 1987; Tomassini *et al.*, 2008; Jones and Forrest, 1995], des problèmes "one-max" ou "long $k$-path" [Horn *et al.*, 1994].

L'apprentissage *au cours du processus* d'optimisation et l'application "en-ligne" (en marche) des informations apprises demeure un problème difficile. Pour y aboutir, un processus d'optimisation doit apprendre à prendre de meilleures décisions locales (e.g. le choix du voisin suivant à un instant donné) en utilisant uniquement des informations globales, acquises le long de la recherche. Pour surmonter ce type de difficultés, l'intégration d'une phase d'apprentissage dans le processus d'optimisation ("learning while optimizing") semble très prometteuse. Notre approche, utilisant des idées de réactivité [Battiti *et al.*, 2008], vise à développer un algorithme capable de se guider et de s'orienter tout seul dans l'espace de solutions.

## 3.2 La recherche locale typique et la vision globale

Particulièrement dans le contexte des heuristiques à base de recherche locale, un risque important est le fait que le processus de recherche pourrait être toujours attiré par les mêmes optima locaux (e.g. des forts "attracteurs"). Tandis qu'il y a de nombreuses méthodes bien étudiées pour aider une recherche à échapper à tout optimum local individuel (e.g. la recherche Tabou a été conçue pour cela), il semble plus difficile de l'empêcher de boucler entre *seulement quelques* optima locaux, bassins d'attraction et plateaux. Rappelons qu'une recherche locale typique se déplace d'une configuration à l'autre sans enregistrer beaucoup de données sur les régions visitées. Habituellement, au moment d'une itération donnée, il n'y a aucune information si la recherche est en train d'explorer une région complètement inconnue (jamais visitée) ou une région déjà explorée (avec des configurations visitées dans la proximité).

En effet, un algorithme typique de recherche locale n'a pas une vue d'ensemble sur son propre chemin d'exploration à travers l'espace, i.e. il ne prend pas en compte les relations

entre des configurations visitées à des *moments différents* dans une longue recherche. De plus, les études qui analysent la structure des espaces de recherche (voir [Hertz *et al.*, 1994; Hamiez and Hao, 2004; Culberson and Gent, 2001] pour la coloration de graphe) sont souvent orientées vers des informations théoriques plutôt que sur leur application dans un algorithme.

Étant donné un processus de recherche traversant l'espace des solutions, quelques questions importantes pourraient être posées :

- à quoi ressemble son chemin d'exploration ?
- quelles régions seront le plus souvent explorées ?
- le processus de recherche, explore-t-il beaucoup plus que quelques régions ?
- quelle est la distribution spatiale des meilleures configurations visitées ?
- ces meilleures configurations sont-elles aléatoirement dispersées ?
- le processus de recherche, peut-il être guidé vers un optimum global ?

**L'hypothèse de clusterisation** En fait, l'étude qui suit est consacrée principalement à ces questions. En utilisant une mesure de distance dans l'espace de recherche (voir Section 3.3.1.1 ci-dessous, ou Chapitre 6 pour une description détaillé), nous définissons la notion de *sphère* : l'ensemble de configurations situées à moins d' une certaine distance (le rayon) d'une configuration centrale. L'hypothèse de clusterisation est la suivante : les optima locaux découverts par la recherche ne sont pas aléatoirement dispersés dans l'espace, mais ils forment des clusters de points qui peuvent être confinés dans *des sphères* de diamètre spécifique.

Dans ce chapitre, nous considérons la recherche locale de la Section 2.2, i.e. l'algorithme Tabou de base *sans critère d'aspiration* mais *avec* une liste Tabou réactive. Par souci de lisibilité, nous rappelons très brièvement dans ce paragraphe ses composantes et sa construction. Essentiellement, la recherche Tabou se déplace itérativement d'une coloration à l'autre en modifiant la couleur d'un sommet en conflit jusqu'à ce qu'une coloration légale soit trouvée, ou qu'une condition d'arrêt soit atteinte. Chaque mouvement effectué (i.e. chaque nouveau changement de couleur) est marqué Tabou pour un certain nombre d'itérations, i.e. la durée Tabou $T_\ell$.

Cet algorithme est en effet capable d'éviter un optimum local indépendant (et même des plateaux de taille raisonnable) avec une simple liste Tabou. Cependant, les questions ci-dessus sont toujours ouvertes, et restent essentielles. Y a-t-il une garantie que l'algorithme explore plus de régions en une semaine qu'en une heure ? Malheureusement, comme pour la plupart des algorithmes de recherche locale, la réponse est *Non !*. Plusieurs tests expérimentaux dans d'autres articles prouvent que les résultats ne peuvent pas être améliorés en augmentant la limite de temps au-delà d'un certain seuil (i.e. plusieurs heures pour la coloration). On peut vérifier dans [Blöchliger and Zufferey, 2008; Hertz *et al.*, 2008] que l'amélioration de performance réalisée grâce à l'augmentation de temps de calcul (en passant d'une heure à dix !) est tout à fait limitée ; de plus, nous doutons qu'en employant 100 ou 1000 heures ce type d'algorithmes Tabou puissent atteindre de nombreuses nouvelles bornes.

## 3.3 Cartographie de l'espace de recherche

Dans cette section, nous explorons la distribution spatiale des meilleures configurations trouvées par notre algorithme de recherche locale. Deux échantillons différents de configurations sont considérés : (i) les meilleurs optima $C_1^*, C_2^*, \ldots$ découverts par plusieurs recherches *indépendantes*, (ii) les meilleures configurations visitées par Tabou en une seule exécution. Nous considérons chaque échantillon comme un ensemble de points dans l'espace de recherche $\Omega$ (un espace $|V|$-dimensionnel), et nous mesurons la distance entre chaque paire de points avec la distance suivante : le nombre minimum de transitions dans le voisinage pour aller d'un point à l'autre, voir Section 3.3.1.1 ci-dessous – la théorie des ensembles nous fournit la distance de transfert entre partitions (voir Chapitre 6), qui est très adaptée à cette situation.

D'abord, nous montrons plusieurs visualisations 3D très intuitives : les points de l'espace $|V|$-dimensionnel $\Omega$ sont plongés dans l'espace euclidien 3D, de façon à minimiser une mesure de distorsion. Nous réalisons cette transformation de $|V|$ dimensions à 3 dimensions avec une procédure de Multidimensional Scaling (MDS) utilisée couramment en l'analyse de données. Puis, nous analysons les valeurs des distances entre tous les points et nous fournissons des arguments en faveur de leur regroupement en clusters dans l'espace de recherche $\Omega$. Nous effectuons également une évaluation du diamètre des clusters et nous montrons que le diamètre ne dépend pas étroitement du type de graphe, mais principalement de $|V|$. Notre hypothèse est que ces clusters peuvent être généralement confinés dans des sphères de rayon $\frac{1}{10}|V|$.

### 3.3.1 L'opération de cartographie

#### 3.3.1.1 Mesure de distance – nombre minimum de transitions dans le voisinage

Considérons un problème d'optimisation combinatoire, avec un espace de recherche $\Omega$ et un voisinage $N$. Pour visualiser correctement les positions d'un échantillon de configurations, nous avons besoin d'une mesure de distance reflétant une longueur selon ce voisinage $N$. Pour illustration, un pas (transition de voisinage) ne doit pas "sauter" une longue distance ; la distance minimum non-nulle doit être 1, correspondant à des configurations voisines. La distance de voisinage est la suivante : le nombre minimum de transitions de voisinage nécessaires pour arriver d'une configuration à l'autre. Étant données deux configurations $C, C' \in \Omega$, la distance de voisinage peut être formellement définie comme le nombre minimum $n$ pour lequel il existe $C_0, C_1, \ldots C_n \in \Omega$ tels que : $C_0 = C, C_n = C'$ et $C_{i+1} \in N(C_i) \forall i \in [0 \ldots n-1]$

Heureusement, dans le cadre de coloration, la distance de transfert entre partitions peut bien fonctionner comme une distance de voisinage. En utilisant la définition de coloration à base de partition (voir Définition 1.3, p. 16), la distance de transfert entre deux colorations $C$ et $C'$ est le nombre minimal de sommets qui doivent être transférés entre les classes de $C$, de sorte que la partition résultante soit égale à $C'$. Comme le transfert d'un sommet d'une classe à l'autre est équivalent à un changement de couleur, cette distance de transfert indique en effet le nombre minimum de pas entre $C$ et $C'$. Rappelons que, dans cette

thèse, nous utilisons une fonction basique de voisinage : les voisins d'une coloration sont essentiellement obtenus en effectuant un changement de couleur (voir Section 2.2.1 pour la définition formelle).

De plus, nous disposons déjà d'une méthode de calcul bien étudiée pour déterminer la distance de transfert. Cette méthode utilise une réduction au problème d'affectation, qui est résolu par la méthode hongroise – voir Section 4.2.2.1. La description formelle et complète de la distance de transfert est disponible au Chapitre 6, mais dans ce qui suit il suffit de dire que $d(C, C')$ représente le nombre minimum de pas (i.e. changements de couleurs) nécessaires pour aller de $C$ à $C'$. Une cartographie très similaire pourrait être effectuée pour d'autres problèmes, avec d'autres distances, associées à des voisinages spécifiques (voir des exemples dans la section 4.5).

#### 3.3.1.2 Multidimensional scaling

Le "Multidimensional Scaling" (MDS, ou positionnement multidimensionnel) est une méthode utilisée couramment en la visualisation de données pour représenter des similitudes ou des dissimilarités dans les données. À partir d'une matrice de distances (e.g. dissimilarités entre des points dans un espace multidimensionnel), cette procédure a pour objectif de représenter les points dans l'espace euclidien ($\mathbb{R}^2$ ou $\mathbb{R}^3$) le plus fidèlement possible, i.e. tel qu'une fonction de distorsion (stress) soit réduite au minimum – on utilise le *Kruskal stress*. Pour notre procédure de MDS, la démarche expérimentale est composée de trois étapes : collecte de données, projection des données, et vérification de la représentation.

**Etape 1 : La matrice de distances dans l'espace $\Omega$** Etant donné un échantillon de $k$-colorations $\{C_1, C_2, \dots C_p\} \subset \Omega$, dans cette étape on construit d'abord la matrice $D_{p\times p}$ où chaque élément $D_{ij} = d(C_i, C_j)$ représente la distance entre $C_i$ et $C_j$.

**Etape 2 : Génération des coordonnées des points 3D** Pour générer les coordonnées cartésiennes dans l'espace $\mathbb{R}^3$, nous employons l'algorithme `cmdscale` (metric/classical MDS), implémenté dans le langage de programmation R pour l'analyse statistique de données. [1] Les coordonnées ainsi fournies sont employées pour tracer des graphiques 3D (scatter plots) ; cela nous permet aussi de calculer la matrice de distances euclidienne $d_{p\times p}$ entre les points 3D. Plus exactement, un appel de fonction de la forme `cmdscale (D, k=3)` semble très efficace pour fournir les coordonnées 3D, qui sont tracées ensuite avec la fonction `scatterplot3d`. Les distances dans l'espace 3D donnent une bonne image de la distribution spatiale des colorations dans l'espace $\Omega$.

**Etape 3 : Evaluation de la qualité de la représentation** Comme l'isométrie entre la matrice des distances euclidiennes ($d_{p\times p}$) et la matrice des distances initiales $D_{p\times p}$ ne peut pas être exactement satisfaite, la fidélité de la représentation est mesurée avec un indicateur de qualité, basé sur la distorsion. En effet, nous mesurons la distorsion (déformation) avec le "stress" classique proposé par Joseph Kruskal [Kruskal, 1964] :

1. Un logiciel libre de plus en plus populaire, voir `www.r-project.org`. Nous avons utilisé gnuR pour produire tous les graphiques dans cette thèse.

$$s_{\text{fit}} = \sqrt{\frac{\sum_{1 \leq i,j \leq p} (D_{ij} - d_{ij})^2}{\sum_{1 \leq i,j \leq p} D_{ij}^2}}$$

Selon les indications fournies par Kruskal dans son article séminal de MDS [Kruskal, 1964], la représentation MDS est : a) mauvaise si $s_{\text{fit}} > 0.2$, *b*) juste si $s_{\text{fit}} \leq 0.1$, c) bonne si $s_{\text{fit}} \leq 0,05$, d) excellente si $s_{\text{fit}} \leq 0,025$ et e) parfaite si $s_{\text{fit}} = 0$. Dans nos analyses, même si le nombre total de points est très grand, nous ne présentons aucune représentation mauvaise (i.e. avec $s_{\text{fit}} > 0.2$).

### 3.3.2 Distribution spatiale des meilleurs optima

Cette section a pour objectif de fournir une représentation intuitive (une carte) de la distribution spatiale des meilleurs optima locaux trouvés dans l'espace de recherche. Pour le graphe standard *dsjc*250.5 (graphe aléatoire DIMACS assez petit avec 250 sommets), le nombre chromatique est inconnu mais aucun algorithme n'a jamais trouvé de coloration légale avec $k < 28$. Nous considérons $k = 27$ et nous avons essayé de trouver les meilleures $k$-colorations en effectuant des centaines d'exécutions de l'algorithme Tabou de base ; les meilleurs optima locaux que nous avons pu trouver ont 3 conflits. En supposant qu'une coloration avec 3 conflits représente un optimum global, nous avons lancé 350 exécutions indépendantes et, pour chaque exécution, nous avons enregistré uniquement le premier minimum global atteint. La Figure 3.1 trace effectivement ces 350 optima trouvés indépendamment par 350 exécutions de la recherche Tabou.

La Figure 3.1 montre une certaine forme de clusterisation (groupement) de ces optima pour cette instance. On peut identifier quelques petites sphères qui pourraient couvrir tous les points – ces sphères représentent les régions qui attirent habituellement l'algorithme de TS (i.e. les principaux "attracteurs"). L'histogramme dans la figure de droite montre la distribution des valeurs de distance entre chaque paire d'optima dans l'espace $\Omega$. La plupart des distances sont soit très petites (inférieures à $\frac{1}{10}|V| = 25$), soit très grandes ; les petites valeurs correspondent à des distances intra-cluster, tandis que les grandes distances correspondent aux distances inter-cluster.

Il est important de mentionner que ce type de tests expérimentaux a besoin de *configurations très rares*, d'excellente qualité. Il aurait été beaucoup plus facile de collectionner des configurations largement distribuées, mais toute projection 3D serait moins concluante et moins pertinente. En effet, les solutions du même graphe (*dsjc*250.5) avec $k = 28$ couleurs sont distribuées plus uniformément, et ainsi sont les solutions de plusieurs instances. Ce simple test expérimental a exigé *plusieurs mois de calcul* sur un processeur 2.8GHz, parce qu'une recherche Tabou a besoin de plusieurs jours pour trouver une solution de cette qualité (i.e. une coloration avec 3 conflits pour $k = 27$).

Finalement, il pourrait être intéressant de noter que les phénomènes observés empiriquement dans cette section pourraient avoir certaines connexions avec des résultats similaires de la communauté de physique statistique. Il existe au moins une étude sur la coloration (voir [Zdeborová and Krzakała, 2007, Fig. 1]) décrivant comment les optima globaux (solutions) sont groupés en petits clusters quand l'instance du problème est très

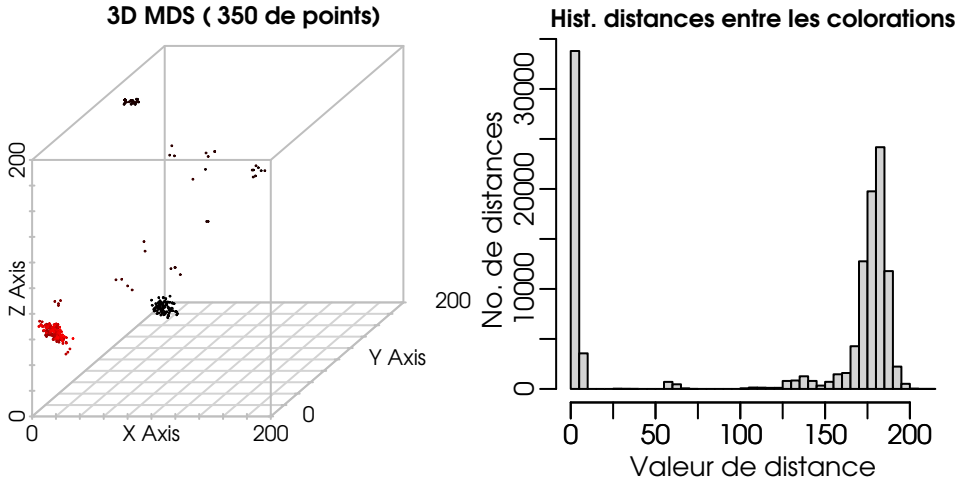


FIGURE 3.1 – 350 optima (indépendamment trouvés) dans l'espace 250-dimensionnel de ($G = dsjc250.5$, $k = 27$) représentés en 3D (stress $s_{\text{fit}} = 0.09$) avec une procédure MDS (graphique de gauche), et l'histogramme des valeurs de distance entre les colorations réelles (graphique de droite). Les points ne sont pas aléatoirement dispersés, mais groupés en clusters, comme confirmé par l'histogramme : les distances entre les colorations sont soit très grandes (distances inter-cluster), soit très petites (distances intra-cluster).

difficile, i.e. à la transition de phase Col/Uncol (Sat/Unsat). En revanche, si l'instance est au-dessous de la transition de phase (des solutions peuvent être facilement atteintes), les optima globaux apparaissent dans un seul cluster "géant". L'instance considérée dans cette section est à la transition de phase, et ainsi, nos clusters pourraient correspondre à ceux tracés dans [Zdeborová and Krzakała, 2007, Fig. 1], voir le cinquième graphique (entre $c_r$ et $c_s$).

### 3.3.3 Distribution spatiale des configurations visitées en série sur une courte période

Dans cette section, nous examinons les colorations visitées par des recherches Tabou courtes., Plus exactement nous nous focalisons uniquement sur les configurations de grande qualité.

**Definition 3.1.** *(Configuration de grande qualité) Une configuration $C \in \Omega$ est une configuration de grande qualité (i.e. profonde, ou difficile–à–trouver) si et seulement si $f(C) \leq B_f$, où $B_f$ est un seuil de qualité (ou d'adaptation). Sinon, $C$ est une configuration de basse ou moyenne qualité.*

Étant donnée une instance $(G, k)$ *ainsi qu'*une première coloration de grande qualité $C_0$, nous lançons l'algorithme Tabou à partir de $C_0$. Le processus de recherche visite une série de colorations et nous notons $C_0, C_1, C_2, \ldots$ les configurations de grande qualité,

satisfaisant $f(C_i) \leq f(C_0)$ – i.e. nous considérons le seuil de qualité $B_f = f(C_0)$. Dans tous nos tests, le nombre de configurations de grande qualité représente seulement une petite fraction du nombre total de colorations visitées le long de la recherche ; nous ignorons les colorations plus mauvaises que $C_0$ parce qu'elles sont largement distribuées et elles peuvent être facilement trouvées. En effet, des configurations de basse qualité peuvent être trouvées plus facilement dans l'espace – i.e. même dans la proximité de l'optimum local initial $C_0$, il devrait y avoir de nombreuses colorations plus mauvaises.

Par exemple, pour l'instance $(le450.25c, 25)$ : (i) il est vraiment difficile de trouver des colorations sans conflits [2], (ii) il est assez facile de trouver des colorations avec 1 conflit en lançant la recherche à partir d'une première coloration à 1 conflit, (iii) de nombreuses colorations à 2 conflits sont dispersées partout autour de la configuration initiale à 1 conflit. Pour ce graphe, et pour une coloration initiale $C_0$ avec un conflit, nous considérons les autres configurations à 1 conflit comme de grande qualité, et les configurations à 2 conflits comme de basse qualité. Si on considère également les configurations de basse qualité dans cette analyse, on trouverait des points très uniformément distribués dans l'espace.

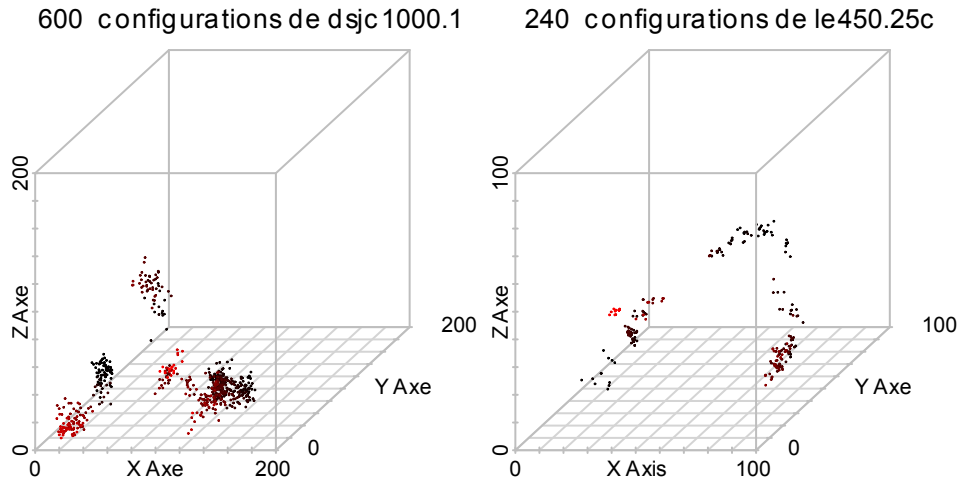


FIGURE 3.2 – Colorations de grande qualité (avec $f(C) \leq 4$) visitées pendant 60000 itérations par la recherche Tabou pour $G = dsjc1000.1$ et $k = 20$ (à gauche) et colorations de grande qualité (avec $f(C) \leq 1$) visitées pendant 25000 itérations par la recherche Tabou pour $G = le450.25.c$ et $k = 25$ (à droite). Les valeurs de stress (déformation) sont $s_{\text{fit}} = 0.19$ et $s_{\text{fit}} = 0.15$, respectivement.

La Figure 3.2 montre des représentations MDS résultant de ce test expérimental. Deux instances sont considérées : (a) le graphe aléatoire $dsjc1000.1$ $(k = 20)$ avec la recherche lancée à partir d'une coloration à 4 conflits (b) le graphe de Leighton $le450.25c$ $(k = 25)$ avec la recherche initiée par une coloration avec 1 conflit. Afin de limiter le nombre de points (et ainsi assurer la fiabilité des représentations MDS), nous considérons chaque série

2. Avant 2008, il y avait un seul algorithme publié [Morgenstern, 1996, p. 353] qui pouvait résoudre $(le450.25,\ k = 25)$.

de colorations $C_0, C_1, C_2, \ldots$ divisée en des intervalles de 100 colorations. Pour faire ces représentations MDS, nous ne considérons que la première coloration de chaque intervalle. Plus exactement, si la recherche visite les configurations de grande qualité suivantes dans cet ordre : $C_0, C_1, C_2, \ldots$, nous traçons seulement l'échantillon $C_0, C_{100}, C_{200}, \ldots$. Les distances entre les colorations visitées à des moments rapprochés (presque consécutivement) sont trop petites – voire nulles, car la recherche fait souvent de petites boucles. Avec un seul représentant par intervalle, il est possible de couvrir une exécution plus longue avec un nombre restreint de points – et ainsi, avec un stress (déformation MDS) limité.

Ces représentations 3D fournissent une bonne image intuitive du chemin d'exploration de la recherche. Dans la Figure 3.2 à gauche, le processus de recherche commence à partir du coin avant-bas-gauche, puis traverse cette section de l'espace jusqu'à ce qu'il atteigne le bord droit ; certaines colorations visitées n'apparaissent pas dans le graphique parce qu'elles sont de basse qualité. A droite dans la même figure, le chemin d'exploration est bien plus clair : il part du coin avant-bas-gauche et se déplace vers le coin avant-bas-droit. En revanche, les clusters sont plus étroits – pour $le450.25c$, les distances entre les clusters sont entre $15\%|V| = 67$ et $22\%|V| = 99$ (voir également la Figure 3.3, le graphique en bas à droite, parce que les graphes de Leighton ont une structure particulière, comme discuté en Section 4.4.4).

### 3.3.4 Distribution spatiale des colorations visitées en série sur une longue période

Cette section est consacrée à une analyse plus formelle (et moins intuitive) d'une *longue série* de colorations de grande qualité visitées par la recherche locale, en considérant plusieurs classes de graphes (Figure 3.3). Nous avons employé un scénario similaire à celui de la Figure 3.2, mais nous examinons beaucoup plus de configurations : en effet, *toutes* les colorations de grande qualité $C_0, C_1, \ldots, C_n$ (i.e. satisfaisant $f(C_i) < f(C_0) \forall i \in [1..n]$, où $n = 40.000$) sont maintenant considérées dans l'analyse. Nous calculons toutes les distances [3] $d(C_i, C_j)$ et, avec l'histogramme des distances, nous examinons le nombre de paires $(C_i, C_j)$ correspondant à chaque valeur de distance.

La Figure 3.3 montre des distributions bimodales des valeurs de distance : il y a de nombreuses distances très petites, et des nombreuses distances très grandes entre les $C_i$'s. Cela confirme l'existence de quelques groupes distants contenant de nombreux $C_i$'s (clusters) : les distances petites correspondent à des couples de colorations à l'intérieur d'un cluster et les grandes distances correspondent à des distances inter-cluster. Si nous notons un "diamètre de cluster" par $C_d$, nous observons que $C_d$ varie de $7\%|V|$ à $10\%|V|$ selon le graphe, tel que :

- il existe de nombreuses paires $(i, j)$ telles que $d(C_i, C_j) < C_d$ ;
- il existe très peu (moins que 1%) de paires $(i, j)$ telles que $C_d < d(C_i, C_j) < 2C_d$ ;
- il existe de nombreuses occurrences de certaines valeurs de distance plus grandes.

Cette distribution statistique des $C_i$'s reflète principalement le chemin du processus de recherche à travers l'espace de recherche – et non pas la distribution spatiale de toutes les

3. En considérant $n = 40000$ colorations, le nombre de distances à calculer n'est pas extrêmement élevé, toutes les $\frac{1}{2} \cdot n(n-1) = 799.980.000$ distances peuvent généralement se calculer en quelques heures.

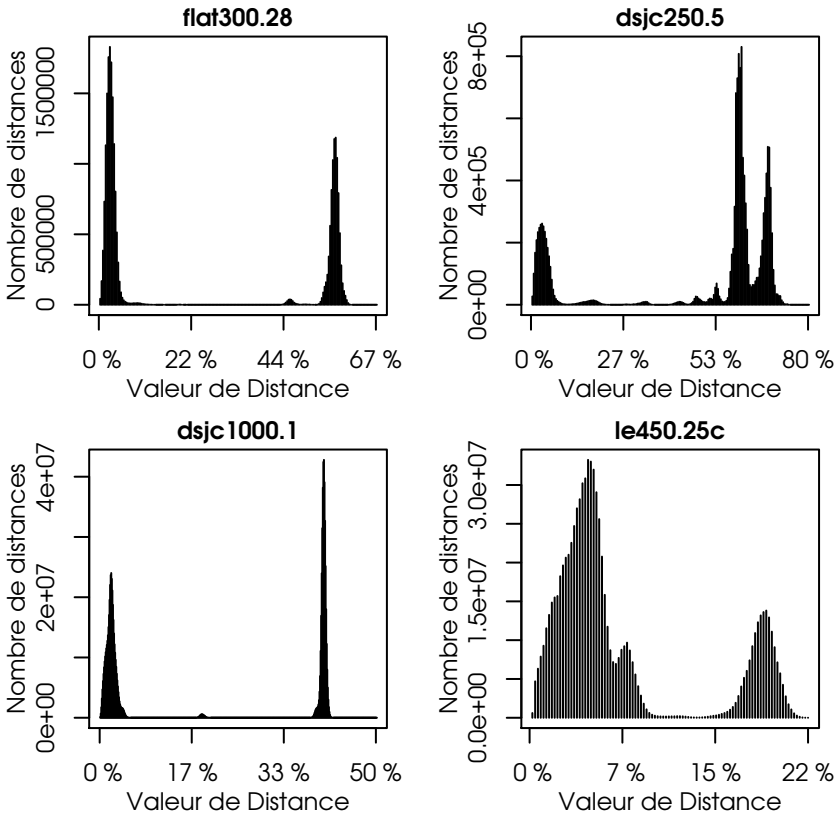


FIGURE 3.3 – Histogrammes des distances entre chaque couple de configurations de grande qualité (notées $C_0, C_1, C_2, \ldots, C_{40000}$) visitées par la recherche Tabou ; $f(C_i)$ est inférieur à : a) $f(C_1) = 4$ pour $G = flat300.28$ et $k = 30$, b) $f(C_1) = 3$ pour $G = dsjc250.5$ et $k = 28$, c) $f(C_1) = 4$ pour $G = dsjc1000.1$ et $k = 20$, et d) $f(C_1) = 1$ pour $G = le450.25c$ et $k = 25$.

configurations de grande qualité existantes. Toutefois, concernant les colorations visitées par la recherche Tabou, il est important de noter *l'hypothèse de clusterisation* : les configurations de grande qualité sont groupées en clusters ; de plus, deux configurations distantes de plus de $\frac{1}{10}|V|$ (la plus grande valeur possible de $C_d$) appartiennent à des *clusters différents*. Nous gardons cette estimation dans le reste du manuscrit et nous l'employons pour proposer des méthodes d'exploration mieux ciblées.

## 3.4 Conclusion du chapitre

Une étude théorique de la structure des clusters est au-delà de la portée de cette analyse. Cela exigerait des algorithmes avec une complexité de calcul trop élevée par rapport aux bénéfices pratiques qu'ils apporteraient. Notre hypothèse simplifiée de clusterisation

est très utile en pratique : il est assez facile d'exploiter le fait que deux configurations de grande qualité distantes de plus de $\frac{1}{10}|V|$ sont dans des clusters différents. On considère que leur "backbone" (affectation de couleurs aux sommets essentiels, non-périphériques) est relativement différente.

Nous supposons que cette hypothèse est satisfaite pour toutes les séries de colorations visitées par la recherche locale Tabou. L'application la plus importante de cette hypothèse est développée dans le Chapitre 4, où nous présentons un algorithme guidé pour assurer la diversification, et un algorithme guidé pour assurer l'intensification. Les deux algorithmes considèrent que l'espace de recherche est organisé dans des sphères de rayon $\frac{1}{10}|V|$. De plus, comme la procédure de recherche Tabou est également utilisée dans l'algorithme évolutionniste Evo–Div du Chapitre 5, la même hypothèse est exploitée afin de garder un écart minimum au sein de la population, i.e. les individus de la population devraient toujours être distants de plus de $\frac{1}{10}|V|$.

# Chapitre 4

# Recherches Locales Guidées : TS-Div et TS-Int

Nous présentons deux algorithmes guidés pour conduire la recherche locale vers certaines régions visées. Les solutions candidates sont supposées structurées en clusters, selon les résultats du Chapitre 3 : les configurations visitées de grande qualité ne sont *pas* aléatoirement dispersées dans l'espace de recherche, mais plutôt groupées en clusters dans des sphères de diamètre $R = \frac{1}{10}|V|$. L'algorithme TS–Div emploie une procédure d'apprentissage pour enregistrer son chemin à travers l'espace de recherche, afin de guider l'exploration vers des *R-sphères* non visitées. Le deuxième algorithme (TS–Int) assure des investigations méticuleuses dans un "périmètre limité", autour d'une configuration donnée. TS–Int emploie un processus de parcours d'arbre pour énumérer toutes les $R$-sphères dans ce périmètre limité, et, chacune de ces sphères est minutieusement explorée par de nombreuses recherches Tabou. Nous avons expérimentalement observé que, si ce périmètre limité contient un optimum global, TS–Int le trouve systématiquement. TS–Div assure la diversité, TS–Int impose l'intensification, et, ensemble ils ont atteint de très bons résultats : ils ont coloré pour la première fois le graphe DIMACS *djsc*1000.9 avec $k = 223$ couleurs. Ce chapitre développe des idées d'un article accepté par *Computers & Operations Research* [Porumbel *et al.*, 2010].

## Sommaire

**4.1 Introduction . . . . . 54**
4.1.1 Motivation et objectifs . . . . . 54
**4.2 TS–Div : recherche continue de régions inconnues . . . . . 55**
4.2.1 Description formelle de TS–Div . . . . . 56
4.2.2 Vitesse de TS–Div . . . . . 59
**4.3 TS–Int : un parcours en largeur de l'espace des solutions . . . 62**

4.3.1 Cartographie par MDS du parcours TS–Int . . . . . 63
**4.4 Résultats et discussions . . . . . 65**
4.4.1 Procédé expérimental . . . . . 65
4.4.2 Résultats standards de TS–Div et de TS–Int . . . . . 66
4.4.3 TS–Int – localisation exacte d'une solution à partir d'une localisation approximative . . . . . 68
4.4.4 La structuration des graphes et du paysage de recherche . . . . . 68
4.4.5 Temps d'exécution de TS–Div et de TS–Int . . . . . 70
**4.5 Vers des applications à d'autres problèmes d'optimisation combinatoire . . . . . 70**
**4.6 Conclusions de chapitre . . . . . 72**

## 4.1 Introduction

### 4.1.1 Motivation et objectifs

La recherche locale classique dispose de peu de mécanismes globaux et, par rapport aux algorithmes évolutionnistes ou à l'optimisation en essaim, elle manque souvent de vision d'ensemble. En effet, toutes les décisions prises par une recherche locale typique sont souvent fondées uniquement sur des informations au niveau microscopique (i.e. le voisinage de la configuration courante, et/ou une histoire courte), sans vision de niveau macroscopique. Par conséquent, ces algorithmes risquent de ne pas couvrir correctement l'espace de recherche, i.e. ils peuvent visiter les mêmes régions à plusieurs reprises, ou ils peuvent simplement rester bloqués dans des bassins d'attraction.

Ce chapitre est centré sur des stratégies qui intègrent des informations macroscopiques sur l'espace de recherche afin de guider le processus de recherche. Ainsi, nous allons d'abord présenter synthétiquement les principes de deux nouveaux algorithmes : TS–Div et TS–Int. Notons qu'une notion centrale dans les deux cas est la *sphère* introduite dans le chapitre précédent : l'ensemble de configurations situées à moins d'une distance $R$ (rayon) d'une configuration centre.

Le premier algorithme (TS–Div) est fondé sur un processus de recherche Tabou et sur une composante d'apprentissage qui mémorise les sphères visitées, afin de guider ensuite le processus de recherche. Comme la mémorisation d'une sphère nécessite uniquement l'enregistrement d'une seule configuration (le centre), nous avons expérimentalement montré que, bien que le nombre de configurations visitées soit toujours très grand, le nombre de sphères visitées peut rester dans des limites très raisonnables (voir également la Figure 1.1, p. 12). Afin de guider le processus de recherche vers des sphères encore inconnues, la composante d'apprentissage conserve en permanence la distance $R$ de tout centre enregistré.

TS–Int (Section 4.3) est centré sur l'exploitation d'un périmètre limité très prometteur. À partir d'une configuration d'entrée, TS–Int effectue des investigations méticuleuses dans sa proximité. D'abord, la sphère de la configuration d'entrée est soumise à une phase d'investigation qui lance plusieurs processus Tabou, autorisés à explorer seulement dans

cette sphère. Chaque processus Tabou peut arriver à un “point de sortie de sphère”, qui est enregistré par TS–Int ; des processus Tabou sont lancés ensuite jusqu'à ce qu'il ne soit plus possible de trouver des “points de sortie” suffisamment distants. Lorsque cette condition est satisfaite, la sphère est considérée “vérifiée” (sans optima globaux). Ensuite, la phase d'investigation de sphère est répétée avec les sphères des “points de sortie”, i.e. chaque “point de sortie” découvert devient le centre d'une nouvelle sphère à vérifier ultérieurement. Les centres de ces sphères sont enregistrés dans une file d'attente ordonnée, et ainsi, les sphères sont traitées dans l'ordre de leur qualité.

Tous les tests expérimentaux fournis dans ce chapitre font appel à un processus explorateur fondé sur un algorithme Tabou de coloration – voir Section 2.2. Tandis que les connaissances spécifiques du problème peuvent toujours être essentielles pour atteindre de meilleurs résultats pratiques, les idées principales de TS–Div et de TS–Int sont indépendantes du concept de coloration. Cependant, grâce à l'utilisation des informations macroscopiques, les recherche locale présentés dans ce chapitre peuvent effectivement concurrencer des algorithmes évolutionnistes plus complexes et plus raffinés.

Le reste du chapitre est organisé comme suit. Dans la Section 4.2 et la Section 4.3, nous présentons ces deux algorithmes : TS–Div (assurant la diversification) et TS–Int (assurant l'intensification). Des résultats, suivis d'une discussion sont présentés en Section 4.4.

## 4.2 TS–Div : recherche continue de régions inconnues

La recherche Tabou classique est très populaire parce qu'elle est capable d'explorer l'espace de solutions sans revisiter inutilement les configurations visitées récemment. Toutefois, les configurations visitées dans un passé éloigné peuvent être revisitées à plusieurs reprises au cours d'une longue recherche. Pour illustration, s'il y a quelques optima locaux avec de grands bassins d'attraction, ceux-ci peuvent toujours attirer la recherche de façon à monopoliser le temps d'exécution. Alors qu'il y a de nombreuses méthodes pour aider une recherche locale à échapper à un optimum local individuel, il semble plus difficile d'éviter de cycler entre quelques bassins d'attraction. C'est l'une des raisons principales pour lesquelles, après un certain seuil de temps, on n'observe plus de progrès important en termes de qualité de résultats.

L'algorithme de TS–Div intègre une composante d'apprentissage pour mémoriser les régions visitées et pour éviter de cycler entre elles. TS–Div utilise une liste Tabou augmentée quand il détecte qu'il arrive sur des configurations appartenant à des sphères déjà explorées. Cette stratégie décourage l'algorithme de revisiter ces sphères, car une liste Tabou augmentée déclenche plus de diversification.

La Figure 4.1 illustre ce principe. Alors qu'il n'est pas possible d'enregistrer toutes les configurations visitées par une recherche locale, il est en revanche possible d'enregistrer les sphères visitées. En effet, ces sphères sont nettement moins nombreuses. Lorsque TS–Div arrive sur une sphère déjà-visitée, la liste de Tabou est augmentée afin de déclencher la diversification, i.e. pour que le processus de recherche abandonne plus vite la sphère courante.

D'une façon générale, on peut considérer les sphères enregistrées comme des *sphères*

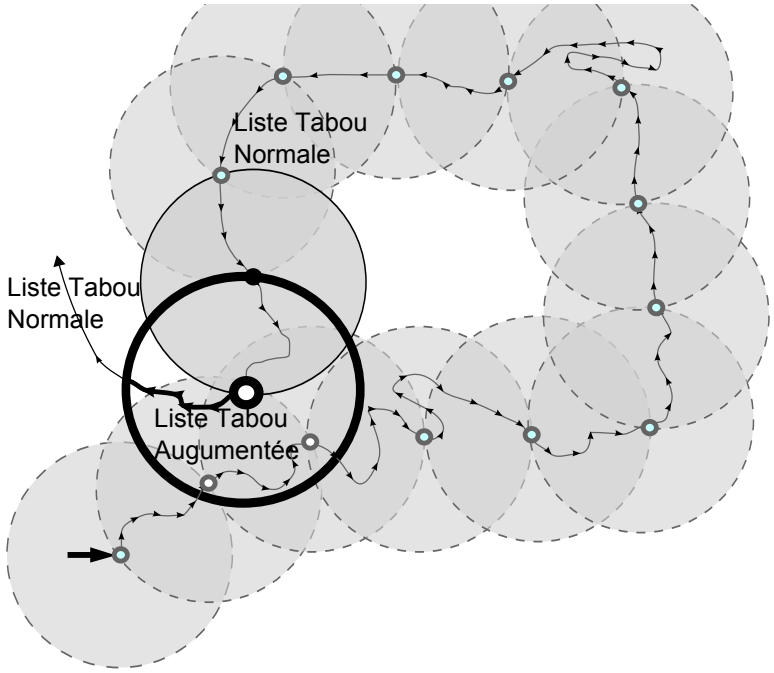


FIGURE 4.1 – Exemple schématique d'exécution de TS–Div. La ligne continue représente le chemin de recherche à travers l'espace. Le centre de la sphère épaisse appartient à une sphère déjà-visitée et TS–Div réalise qu'il a besoin de diversifier ; dans cette sphère, il utilise une liste Tabou plus longue, et ainsi, une diversification plus forte est introduite dans le processus de recherche.

*Tabou* – ou des "sphères déconseillées", que le processus de recherche est découragé à (re-)visiter. La recherche Tabou interdit complètement certaines configurations pour une période *courte* mais elle n'a pas été conçue pour imposer des statuts Tabou définitifs. Cela peut se réaliser uniquement avec des opérations similaires à un niveau macroscopique, i.e. en enregistrant et en interdisant des régions au lieu des configurations. Dans un certain sens, on peut dire que TS–Div est un *Algorithme Tabou À Un Niveau de Granularité Supérieur*.

### 4.2.1 Description formelle de TS–Div

TS–Div (voir Algorithme 4.1) est basé sur deux processus : (i) le processus exploratoire Tabou classique, (ii) le processus réactif d'apprentissage, voir Pas `4.(b)-(d)` de l'Algorithme 4.1. Les principaux outils de la partie apprentissage sont la sphère et la mesure de distance, qui seront détaillées dans ce qui suit.

Pour le problème de coloration, nous employons la distance de transfert entre partitions (voir Section 4.2.2.1) ; sa méthode de calcul est décrite en détail au Chapitre 6, mais dans ce chapitre il suffit de dire que $d(C_a, C_b)$ représente une mesure de type "plus courte

chaîne de transitions de voisinage" pour relier $C_a$ à $C_b$. Plus formellement, cette distance est le minimum nombre $n$ tel qu'il existe $C_0, C_1, \dots C_n \in \Omega$ avec $C_0 = C_a, C_n = C_b$ et $C_{i+1} \in N(C_i)$ $(\forall i \in [0 \dots n-1])$. Des distances de type "plus court chemin via des transitions de voisinage" peuvent être définies pour d'autres problèmes (voir Section 4.5). Étant donnée une telle distance, la $R$-sphère de $C$ est définie comme la sphère fermée de rayon $R$ centrée en $C$ :

**Definition 4.1.** *(Sphère) Étant donné une configuration (un centre) $C \in \Omega$ et un rayon $R \in \mathbb{N}^*$, la R-sphère $\mathscr{S}_R(C)$ est l'ensemble des configurations $C' \in \Omega$ telles que $d(C, C') \leq R$.*

Selon l'hypothèse de clusterisation (Section 3.3), nous utilisons uniquement des $R$-sphères de rayon $R = \frac{1}{10}|V|$ dans le reste du chapitre. Si l'indice $R$ de $\mathscr{S}_R$ n'est pas présent, alors nous parlons d'une "$R$-sphère" avec $R = \frac{1}{10}|V|$. Deux configurations $C_a$ et $C_b$ telles que $d(C_a, C_b) \leq R = \frac{1}{10}|V|$ sont *proches* ou *connexes* ; sinon, elles sont *distinctes* ou *$R$-distantes*. Si $d(C_a, C_b) > \frac{|V|}{2}$, nous disons que $C_a$ et $C_b$ sont *complètement différentes*.

Au début, TS–Div (Algorithme 4.1) effectue les mêmes mouvements que l'algorithme de base, mais il enregistre les centres des sphères visitées. À une itération donnée, notons $C_p$ le dernier centre enregistré (de la sphère courante) et notons $C$ la configuration courante. La première tâche de la composante d'apprentissage est de calculer $d(C, C_p)$ pour savoir si $C$ est toujours dans la sphère $\mathscr{S}(C_p)$ – voir Pas `4.(b)` de l'Algorithme 4.1. Tant que $C \in \mathscr{S}(C_p)$, le processus de recherche est toujours dans la sphère de $C_p$ et TS–Div fait essentiellement les mêmes mouvements que la recherche Tabou de base. Nous disons que le processus de recherche est en train de pivoter autour du pivot $C_p$.

Dès que la recherche sort de la sphère courante (dès que $C \notin \mathscr{S}(C_p)$), la composante d'apprentissage se focalise sur des décisions de guidage. Elle compare d'abord $C$ à l'archive des configurations (centres) enregistrées, voir procédure `Already-Visited` appelée dans le Pas `4.(b).ii` de l'Algorithme 4.1. Avec cette procédure, TS–Div vérifie si $C$ fait partie d'une sphère déjà visitée. Si ce n'est pas le cas, alors $C$ représente le centre d'une nouvelle sphère ; la composante d'apprentissage enregistre ce nouveau centre, mais le processus d'exploration continue normalement. Autrement, si $C$ fait partie d'une sphère déjà visitée, la composante d'apprentissage intervient dans le processus de recherche : une phase de diversification est nécessaire. À cette fin, elle augmente la durée Tabou $T_\ell$ avec une valeur $T_{\text{aug}}$, comme nous le détaillons dans la prochaine section.

**Diversification via l'augmentation de durée Tabou** La longueur de la liste Tabou (ou durée Tabou) fournit un mécanisme simple de diversification qui est déjà connu dans la littérature [Battiti *et al.*, 2008]. Rappelons (Section 2.4) que l'effet principal produit par la liste Tabou est que la recherche peut choisir uniquement des mouvements qui n'ont *pas* été exécutés durant les dernières $T_\ell + T_{\text{aug}}$ itérations. Avec une liste Tabou plus longue, le processus est forcé de sélectionner des mouvements moins répétitifs et plus divers – les derniers $T_\ell + T_{\text{aug}}$ mouvements ne peuvent pas être répétés. D'autre part, une liste Tabou plus courte détermine une intensification plus forte ; il est plus facile de revenir à des configurations précédemment explorées, en re-exécutant des mouvements faits dans un passé proche.

**Algorithme 4.1** : La (méta) heuristique TS–Div

```
PROCÉDURE ALREADY-VISITED
Entrée : configuration courante C
Valeur de retour : VRAI ou FAUX
 1. Forall configurations enregistrées C_rec :
    - Si d(C, C_rec) ≤ R
        Retour VRAI
 2. Retour FAUX

ALGORITHME TABUSEARCH-DIV
Entrée : l'espace de recherche Ω
Valeur de retour : C_best, la meilleure configuration jamais visitée
C : la configuration courante
DÉBUT
 1. C   = configuration aléatoire de Ω
 2. C_p  = C            /*le pivot, i.e. dernier centre de sphère enregistré*/
 3. T_aug = 0           /*l'augmentation de liste Tabou déclenchée par TS-Div*/
 4. Tant que condition d'arrêt non atteinte
    (a) C = le meilleur voisin non-Tabou en N(C)
    (b) Si d(C, Cp) > R
        i. C_p = C
       ii. Si ALREADY-VISITED(C_p) Alors
             - Incrémenter T_aug
           Sinon
             - T_aug = 0
             - Enregistrer C_p
    (c) Marquer C Tabou pour T_ℓ + T_aug itérations
                  /*T_ℓ =durée interne utilisée par l'Alg. Tabou de base*/
                /*T_aug = durée de diversification induite par l'alg. TS-Div*/
    (d) Si (f(C) < f(C_p))
        - remplacer C_p avec C dans l'archive
        - C_p = C                    /*i.e. "re-centrer" la sphère courante*/
    (e) Si (f(C) < f(C_best))
        - C_best = C
 5. Renvoyer C_best
FIN
```

Pour récapituler, la variation de la durée Tabou (via le facteur $T_{\text{aug}}$) contrôle la balance entre la diversification et l'intensification : plus grande est la valeur $T_{\text{aug}}$, plus il y a de la diversification. De cette façon, un contrôle approprié de $T_{\text{aug}}$ garantit que TS–Div découvre de nouvelles régions à tout moment. En effet, notre réglage de $T_{\text{aug}}$ garantit que le processus ne peut pas se coincer en (ré)explorant uniquement des sphères déjà visitées. Si cela arrivait, la liste Tabou pourrait croître indéfiniment – $T_{\text{aug}}$ ne décroît que lorsque le processus de recherche trouve une nouvelle sphère. Une valeur suffisamment élevée de $T_l + T_{\text{aug}}$ sera capable de diversifier (plus tôt ou plus tard) et d'arrêter tout bouclage entre des sphères déjà-visitées.

Notons qu'il existe de nombreuses manières de créer de la diversité au moment où TS–Div détecte qu'il re-visite une sphère. Par exemple, on aurait pu simplement appliquer un opérateur de marche aléatoire, ou une perturbation classique de recherche locale itérée.

### 4.2.2 Vitesse de TS–Div

Le seul aspect problématique de TS–Div concerne le ralentissement introduit par la composante d'apprentissage qui doit calculer de nombreuses distances. Notre objectif est de maintenir le temps consommé par la composante d'apprentissage dans le même ordre de grandeur que le temps consommé par le processus exploratoire. Pour cela, nous nous concentrons sur deux facteurs cruciaux : le temps de calcul de la distance et le nombre de distances calculées. Si un de ceux deux éléments n'est pas conservé dans des limites raisonnables, la vitesse de TS–Div est compromise.

#### 4.2.2.1 Définition et calcul rapide de la distance

En utilisant la représentation à base de partitions (Définition 1.3, p. 16), la distance entre deux colorations $C_a$ et $C_b$ est le nombre minimal de sommets qui doivent être transférés d'une classe à l'autre en $C_a$ (qui doivent changer la couleur), afin que la partition résultante soit égale à $C_b$. Cela est équivalent au nombre minimal de mouvements (changements de couleurs) que notre recherche Tabou doit faire pour arriver de $C_a$ à $C_b$. Il existe déjà une méthode de calcul publiée dans la littérature – voir [Gusfield, 2002] pour une approche générale dans la théorie des ensembles ou [Glass and Pruegel-Bennett, 2005] pour une application à la coloration. Cette méthode réduit le calcul à un problème d'affectation qui peut être résolu par l'algorithme hongrois ; la complexité totale est de $O(|V| + k^3)$. Toutefois, sous certaines conditions (voir le Chapitre 6), il est possible de faire ce calcul en $O(|V|)$ avec une méthode spécifique.

En principe, la distance est calculée avec la formule $d(C_a, C_b) = |V| - s(C_a, C_b)$, où $s$ est une mesure de similarité définie comme suit. Utilisant les notations de la Section 1.2.2 (e.g. la classe de couleur $i$ de $C$ est notée $C^i$), $s(C_a, C_b)$ est égal à $\max_{\sigma \in \Pi} \sum_{1 \leq i \leq k} T_{i,\sigma(i)}$, où $\Pi$ est l'ensemble de toutes les permutations de $\{1, 2, \ldots, k\}$ et $T$ est une matrice $k \times k$ telle que $T_{ij} = |C_a^i \cap C_b^j|$ [Gusfield, 2002; Glass and Pruegel-Bennett, 2005]. Cette somme peut être considérée comme un problème classique d'affectation et ainsi être calculée avec l'algorithme hongrois. Cependant, nous n'avons appliqué l'algorithme hongrois complet que très rarement (moins que 5% des cas) parce que le calcul peut être simplifié en exploitant quelques particularités du problème. La matrice $T$ contient au maximum $|V|$ éléments non nuls et, comme indiqué aussi dans [Gusfield, 2002, §2], ils peuvent être remplis en $O(|V|)$ en utilisant une structure de données appropriée. En effet, les seuls éléments non nuls de $T$ sont tous situés aux positions $T_{C_a(x),C_b(x)}$ (avec $x \in V$) et notre algorithme manipule seulement ces éléments.

De plus, TS–Div n'exige pas une valeur précise de la distance ; notre algorithme de calcul doit seulement décider si $d(C_a, C_b) > R$ – ou $s(C_a, C_b) < |V| - R$. Comme $s(C_a, C_b)$ est toujours inférieur à $s'(C_a, C_b) = \sum_{1 \leq i \leq k} \max_j T_{ij}$ (parce que $T_{i,\sigma(i)} \leq \max_j T_{ij}, \forall \sigma \in \Pi$), il a été très souvent suffisant de vérifier que $s'(C_a, C_b) < |V| - R$ pour décider que $s(C_a, C_b) < |V| - R$. Tous les maxima $\max_j T_{ij}$ peuvent être identifiés en parcourant les éléments non nuls de $T$, et ainsi, $s'(C_a, C_b)$ peut être calculé en $O(|V|)$. D'autres conditions qui peuvent simplifier le calcul sont détaillées dans le Chapitre 6.

En résumé, cet algorithme de calcul de distance exige (en moyenne) approximativement

le même temps de calcul qu'une itération de la recherche Tabou : une itération a besoin de $O(|V| + |CE| \times k) > O(|V|)$. La prochaine section décrit une méthode pour maintenir le nombre d'itérations et le nombre de calculs de la distance dans le même ordre de grandeur pendant de longues exécutions de TS–Div. Ainsi, notre procédure de calcul de distance garantit que le ralentissement provoqué par la composante d'apprentissage peut rester dans des limites acceptables.

#### 4.2.2.2 Nombre de calculs de la distance

Concernant le nombre de calculs de distances, l'opération de parcours de toute l'archive (i.e. dans la routine `Already-Visited`, voir Algorithme 4.1) est la plus critique : TS–Div doit calculer la distance entre la coloration courante et *toutes* les colorations stockées dans l'archive (voir la boucle `Forall`, Pas `1`, routine `Already-Visited`). Notre objectif est de maintenir le temps consommé par la composante d'apprentissage dans le même ordre de grandeur que le temps consommé par la composante exploratoire. Si la taille d'archive dépasse une certaine limite, le nombre de calculs de la distance devient trop grand, compromettant la vitesse. Cependant, si nous focalisons la composante d'apprentissage sur les configurations de grande qualité, l'exploration de l'archive peut devenir gérable. Ainsi, nous permettons à la routine ci-dessus de parcourir *uniquement* les configurations de grande qualité, dans les couches "profondes" de l'espace de recherche.

A chaque mouvement, la plupart des recherches heuristiques essayent de réduire la fonction objectif au minimum. Ainsi, l'objectif implicite est de rester plus de temps dans des couches profondes de l'espace de recherche (avec des configurations de qualité) que dans des couches supérieures (avec des configurations moins prometteuses). La probabilité de trouver une solution en recherchant les couches supérieures est plus faible (le nombre d'optima locaux lui-même est limité). Plus important, le risque de cycler est très limité dans ces couches supérieures où il n'y a pas de forts "attracteurs" pour la recherche – voir [Du and Pardalos, 2007, p. 426] pour des discussions sur le modèle multi-couche de l'espace de recherche. Ainsi, nous pouvons laisser le processus d'apprentissage se concentrer uniquement sur les couches plus profondes de l'espace de recherche sans prendre trop de risques.

Le seul détail qui doit être spécifié est un seuil formel pour définir la frontière entre l'espace profond et l'espace supérieur, entre les configurations de grande qualité et de basse qualité. Rappelons (Définition 3.1, p. 48) que nous disons que la configuration $C \in \Omega$ est de grande qualité si et seulement si $f(C) \leq B_f$ ; alors, $B_f$ est notre seuil formel de qualité. Ce seuil peut être modifié par TS–Div en fonction des configurations découvertes jusqu'à un moment donné.

En effet, $B_f$ est automatiquement abaissé et augmenté par TS–Div selon l'équilibre entre le nombre de distances calculées et le nombre d'itérations, afin de s'assurer que le nombre d'itérations reste dans le même ordre de grandeur que le nombre de calculs de la distance. Cela s'est avéré une règle approximative très pratique. Plus précisément, $B_f$ commande directement le temps consommé par la composante d'apprentissage : le Pas `4.(b)` est ainsi exécuté seulement si $f(C) < B_f$. Dans la pratique, $B_f$ varie de 5 conflits à 20 ; pour certaines instances, même $B_f = \infty$ peut conduire à une vitesse d'exécution

acceptable.

Concernant la distance calculée au Pas `4.(b)`, cela exige un seul calcul par itération et ne pose donc pas trop de problèmes. De plus, ce calcul de distance n'est pas toujours nécessaire : si $d(C_p, C) < R$, TS–Div a besoin d'au moins $R - d(C_p, C)$ pas pour sortir de la sphère de $C_p$. Ainsi, après avoir calculé une fois la distance dans ce pas, TS–Div peut l'éviter pendant les prochaines $R - d(C_p, C)$ itérations sans aucun risque. Nous avons expérimentalement observé que plus de 90% de calculs de distance (dans ce pas) peuvent être économisés de cette manière.

Certains détails simples d'optimisation peuvent encore réduire le nombre de calculs de distances. Bien que nous n'ayons pas employé cela pour la coloration de graphe, notons qu'il est possible d'accroître le rayon $R$ pour réduire le nombre de sphères considérées. Un autre mécanisme simple consiste à transformer l'archive en une file d'attente qui enlève l'élément le plus ancien à chaque opération d'insertion. Dans ce cas-là, TS–Div devient une *Recherche Tabou de Niveau Supérieur à Double Liste* : (1) la liste traditionnelle des dernières configurations visitées qui sont interdites, (2) la liste Tabou des sphères, employée pour éviter de revisiter des sphères visitées dans le passé récent. La signification de l'expression "passé récent" dépend de la taille de la file d'attente qui devrait être réglée selon le ralentissement introduit par la composante d'apprentissage.

**Confirmation expérimentale du ralentissement causé par les calculs de distance** Pour mieux présenter l'importance du nombre de calculs de la distance, ce paragraphe montre expérimentalement que le ralentissement de la composante d'apprentissage est acceptable si le nombre de calculs de distance reste dans le même ordre de grandeur que le nombre d'itérations. Comme il n'est pas mathématiquement possible de déterminer le nombre exact de calculs exigés par TS–Div (il dépend de l'évolution de la recherche), nous avons mené une analyse statistique expérimentale sur une instance classique (G=$dsjc250.5$, $k = 27$). Nous avons lancé 10 exécutions indépendantes avec une limite de 16 milliards d'itérations. Nous avons inspecté les ressources utilisées (en moyenne) pour chaque valeur faisable de $B_f$ – dans ce test expérimental, $B_f$ est fixe.

| $B_f(fixe)$ | 0 | 5 | 6 | 7 | **8** | 9 |
|---|---|---|---|---|---|---|
| Nbre. calculs de distance ($\times 10^6$) | 0 | 1.5 | 31.7 | 877.7 | **13743** | 128932 |
| Temps (heures:min) | 120:56 | 120:51 | 120:37 | 122:32 | **149:17** | 312:43 |
| Nbre. configurations dans l'archive | pas archive | 523.8 | 6244.6 | 37810 | **154462** | 602523 |
| Mémoire (MB) | 0 | 0.523 | 6.2 | 37 | **154** | 6025 |

TABLE 4.1 – Les ressources consommées par la recherche Tabou (colonne 2) et par TS–Div (colonnes 3–7) avec plusieurs valeurs fixes de $B_f$, entre 5 et 9 pour ($dsjc250.5$, 27). Toutes les valeurs rapportées représentent la moyenne sur 10 exécutions de $16 \times 10^9$ itérations.

Dans le Tableau 4.1, nous présentons (2ème et 3ème ligne, respectivement) le nombre de calculs des distances et le nombre d'heures exigées (en moyenne) par une exécution

**Algorithme 4.2** : Pseudo-code TS–Int

**Valeur d'entrée :** une configuration de départ $C_s$
**Valeur de retour :** la meilleure configuration jamais visité
$Q = \{C_s\}$
**DÉBUT**
**Tant que** file $Q$ n'est **pas** vide
1. $C_s$ =TOP($Q$)
2. **Lance** un processus Tabou à partir de $C_s$
   **Pour chaque** itération du processus Tabou, soit $C$ la configuration courante :
   (a) **Si** $d(C, C_s) > 2R$ **alors** *stop processus Tabou* et **Go To 3**
   (b) **Si** $d(C, C_s) > R$ **et** $C$ est de **grande qualité,** *stop processus Tabou* et :
      i. **Si** $d(C, C_i) > R$ **pour tous** $C_i \in Q$
         - Enfiler ($C$,$Q$)
3. **Si** la $R$-sphère $\mathscr{S}(C_s)$ de $C_s$ est "vérifiée"
   (a) Défiler ($Q$)
**FIN**

TS–Div. La colonne TS montre que la recherche Tabou de base a besoin d'environ 120 heures pour finir les $16 \times 10^9$ itérations. Si $B_f \leq 6$, le ralentissement de la composante d'apprentissage est négligeable, parce que le nombre de calculs de la distance est trop petit (par rapport aux 16 milliards d'itérations). Pour $B_f \in \{7, 8\}$, la différence de temps est toujours acceptable (i.e. inférieure à un quart du temps de base 120h) mais elle devient trop grande pour $B_f = 9$ – dès que le nombre de calculs de la distance devient trop grand par rapport au nombre d'itérations (128 milliards $\gg$ 16 milliards). La mémoire consommée ($5^{\text{ème}}$ ligne) par l'archive est proportionnelle avec le nombre de configurations stockées dans l'archive ($4^{\text{ème}}$ ligne), car chaque configuration exige $|V| \times 4 \simeq 1000$ octets. [1]

## 4.3 TS–Int : un parcours en largeur de l'espace des solutions

Une difficulté typique des algorithmes de recherche locale est la suivante : la recherche arrive dans un minimum local situé dans la sphère d'une solution, et, il faut choisir les prochains mouvements. Dans le meilleur des cas, la recherche choisit des mouvements dans la direction de la solution et, ainsi, cette solution est découverte rapidement. Mais, il pourrait y avoir des millions de mouvements possibles pour échapper à ce minimum local ; la plupart de ces mouvements pourraient conduire finalement vers des régions sans solutions et il est évidement impossible de savoir cela dès le début. D'ailleurs, en particulier pour TS–Div, si un mouvement est choisi dans une direction différente et TS–Div sort de la sphère contenant la solution, le processus est découragé à y revenir. Pour faire face à cette difficulté, TS–Int est un algorithme qui explore méthodiquement un périmètre limité dans la proximité d'une configuration $C_s$, donnée comme valeur d'entrée.

Ce périmètre limité est divisé implicitement dans des sphères qui sont parcourues par TS–Int (voir Algorithme 4.2) ; pour chaque sphère, TS–Int exécute une étape d'*investigation de sphère* (voir le pas 2 de l'Algorithme 4.2). Lors d'une investigation

1. Les machines modernes utilisent des entiers sur 64 bits, mais un nombre entier (une couleur) peut aussi bien être stocké sur 8 bits (suffisamment, si $k < 2^8$)

d'une sphère, le processus explore minutieusement la $R$-sphère $\mathscr{S}(C_s)$ de $C_s$ en lançant de nombreux *processus Tabou* qui explorent seulement la proximité de $C_s$. Plus exactement, chaque processus Tabou est arrêté quand il atteint une configuration de grande qualité $C$ en dehors de $\mathscr{S}(C_s)$, i.e. un "point de sortie" de la sphère. Si $C$ vérifie certaines conditions (détaillées ci-dessous), elle est enregistrée dans une file d'attente $Q$ représentant des centres des sphères à investiguer plus tard. Après avoir fini une étape d'investigation de sphère (TS–Int a lancé suffisamment de processus pour considérer la sphère "vérifiée" – sans meilleures solutions), TS–Int prend le centre de la suivante sphère de $Q$ (voir Pas `1`, Algorithme 4.2) et répète la même investigation de sphère (l'ancien centre de la sphère est enlevé de Q, voir pas `3.(a)`). Notons que la file $Q$ trie les centres des sphères selon la fonction objectif.

L'investigation d'une sphère est arrêtée dès que la sphère $\mathscr{S}(C_s)$ est considérée "vérifiée" – i.e. aucun minimum global ne pourrait être atteint avec un processus Tabou. Il est essentiel de décider correctement du moment où la sphère peut être déclarée "vérifiée". Comme indiqué dans le paragraphe ci-dessus, un processus Tabou est arrêté lorsqu'il trouve une configuration $C \notin \mathscr{S}(C_s)$ qui est également de grande qualité (i.e. satisfaisant $f(C) < B_f$, comme dans le cas du TS–Div). La configuration $C$ est insérée dans $Q$ seulement si $C$ n'est pas déjà contenue dans une autre sphère de $Q$ (voir le pas `1.(b).i`, Algorithme 4.2). Dans ce cas-ci, nous disons que le processus Tabou a été *réussi* ; autrement, il a été *échoué* – il n'a pas trouvé de nouvelles sphères prometteuses. Notons que nous n'arrêtons pas un processus Tabou dès qu'il sort de $\mathscr{S}(C_s)$, parce qu'il peut entrer et sortir plusieurs fois de la sphère avant de s'éloigner définitivement. Cependant, si un processus Tabou s'éloigne trop du centre sans trouver de solutions de grande qualité (i.e. si $d(C, C_s) > 2R$, voir le pas `2.(a)`), alors le processus de recherche est arrêté et il est également marqué *échoué*.

Avant de déclarer une sphère $\mathscr{S}(C_s)$ comme "vérifiée" (i.e. sans optima globaux) on doit s'assurer que TS–Int lance des processus Tabou à partir de $C_s$ tant qu'il est toujours possible de découvrir de nouveaux points de sortie distants et de grande qualité. Nous utilisons une condition qui stipule que TS–Int lance une première série d'au moins $Y$ (habituellement $Y = 15$) processus Tabou à partir de $C_s$ et, qu'en plus, il lance un certain nombre $X$ (nous employons $X = 10$) de processus Tabou pour chaque *processus réussi* (i.e. pour chaque nouvelle insertion dans $Q$). De cette manière, l'investigation d'une sphère est finie seulement si le nombre de processus échoués est au moins $X = 10$ fois plus grand que le nombre de processus Tabou réussis – cela est suffisamment pour considérer que la sphère a été explorée "sous tous les angles".

### 4.3.1 Cartographie par MDS du parcours TS–Int

TS–Int organise l'espace de recherche dans des sphères qui sont parcourues d'une façon méthodique, via un algorithme de parcours d'arbre (similaire à un parcours en largeur ou $A^*$). La configuration de départ est la racine de l'arbre. Une configuration $C_s$ est reliée par une arête vers tous les nouveaux centres de sphère découverts par des processus Tabou lancés à partir de $C_s$. Chaque nouveau centre de sphère $C$ constitue un nouveau sommet de l'arbre relié à la configuration à partir de laquelle $C$ a été atteint. Le degré de $C_s$ indique

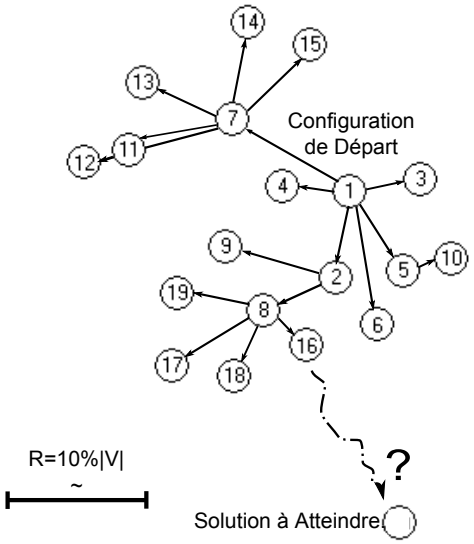


FIGURE 4.2 – Illustration (Multidimensional Scaling) d'une évolution réelle de TS–Int pour ($G = le450.25$, $k = 25$). Tous les points représentent des configurations avec $f = 1$ (dans l'espace $|V|$-dimensionnel). Ils ont été découverts dans l'ordre des nombres : 1 est le point de départ, les points $2, 3, \ldots, 7$ sont découverts par des processus Tabou lancés à partir du point 1, etc. L'indicateur de distorsion (stress) est acceptable, i.e. $s_{\text{fit}} = 0.05$, voir Section 3.3.1.2 (p. 46).

le nombre de nouveaux centres R-distants atteints à partir de $C_s$ – rappelons que tous les centres des sphères enregistrés en $Q$ sont R-distants. Des processus Tabou sont lancés à partir de $C_s$ tant qu'il y a des chances de découvrir de nouveaux centres $R$-distants.

Pour l'illustration, la Figure 4.2 trace une exécution TS–Int simplifiée. La solution est située à distance $23\%|V| < 3R$ du sommet 1 (i.e. la configuration de départ) et elle est toujours atteinte après une exploration d'arbre en profondeur 3 (sur 3 niveaux). Normalement, le degré moyen de cet arbre pour cette instance est environ 20, mais nous avons utilisé des valeurs plus petites de $X$ et $Y$ pour rendre chaque investigation de sphère plus rapide. Dans la réalité, toute solution sur un rayon de $3R$ peut être atteinte par TS–Int après avoir exploré environ $20^3 = 8000$ sommets. Nous avons expérimentalement observé aussi sur d'autres graphes que TS–Int trouve toujours la solution s'il commence à partir d'un point situé à moins de $\frac{1}{4}|V|$ de la solution (voir aussi la Section 4.4.3).

L'algorithme proposé dans cette section (TS–Int) peut être lancé à partir d'une coloration d'entrée unique $C_s$, mais il peut accepter comme entrée une file de configurations *complètement différentes* – i.e. la file $Q$ peut être donnée comme entrée. Pour illustration, il est possible de prendre l'archive enregistrée par TS–Div, de garder uniquement un ensemble des meilleures colorations complètement différentes et de construire une file d'attente pour TS–Int. Ainsi, TS–Int est capable de faire une recherche intensifiée autour des points les plus prometteurs découverts par un autre algorithme. Notons que TS–Int

emploie également le paramètre $B_f$ pour faire la distinction entre des configurations de grande qualité et des configurations de basse qualité. Nous avons toujours pris la valeur fournie à la fin de TS–Div, mais il est possible de régler automatiquement cette valeur, selon le nombre de distances que TS–Int est capable de calculer (d'une manière analogue à TS–Div, voir la Section 4.2.2.2).

## 4.4 Résultats et discussions

Dans cette section, nous évaluons expérimentalement les algorithmes guidés. Le premier, TS–Div, est un algorithme indépendant d'optimisation classique ; TS–Int fonctionne dans une phase de post-optimisation – il peut uniquement améliorer une certaine configuration fournie par TS–Div comme son entrée. Tous les tests expérimentaux sont effectués sur plusieurs graphes difficiles (les plus connus) du benchmark DIMACS (voir Section 1.2.3, p. 17). Nous discutons également de l'influence de la structure de graphe sur le *paysage de recherche* (associé à notre voisinage) et cela nous permet d'expliquer les résultats de plusieurs algorithmes.

### 4.4.1 Procédé expérimental

Tout d'abord, nous devons préciser qu'au cours des premières itérations de TS–Div, cet algorithme est équivalent à l'algorithme Tabou de base, car l'archive de TS–Div est (presque) vide. La composante d'apprentissage intervient dans le processus d'exploration uniquement après un certain nombre d'itérations, i.e. dès que TS–Div tente de revisiter des sphères déjà visitées. Si la recherche Tabou de base est capable de résoudre rapidement une instance, TS–Div ne la résout pas plus rapidement. La contribution de TS–Div concerne le long terme, i.e. il aide la recherche Tabou sur des instances difficiles où elle échouerait autrement.

Pour examiner le comportement de TS–Div, nous effectuons 10 exécutions indépendantes, chacune avec une limite de temps de 50 heures. Dans cette limite de temps, TS–Div réinitialise sa recherche avec une $k$-coloration aléatoire après chaque série de 40 millions d'itérations. Cependant, l'archive des sphères est unique pendant toutes ces initialisations. La statistique des résultats (les moyennes) est fondée sur ces 10 exécutions indépendantes de 50 heures. TS–Int est examiné dans les mêmes conditions expérimentales, i.e. sur 10 exécutions indépendantes, chacune avec une limite de temps de 50 heures. La Section 4.4.5 explique les raisons de l'utilisation de ce temps de fonctionnement, par rapport à la pratique courante dans la littérature.

Concernant TS–Int, rappelons qu'il doit disposer d'une configuration $C_s$ d'entrée : TS–Int recherche la solution dans un périmètre limité autour de $C_s$. De cette façon, le succès de TS–Int dépend entièrement de la distance de $C_s$ à une solution. Nous effectuons d'abord une exécution de TS–Div et nous rassemblons les meilleures colorations stockées dans l'archive à la fin de TS–Div. Ces colorations sont filtrées afin de garder seulement quelques colorations complètement différentes comme points de départ pour TS–Int ; nous présentons les résultats de TS-Int sur les meilleures colorations d'entrée (celles qui conduisent TS-Int aux meilleurs résultats).

### 4.4.2 Résultats standards de TS–Div et de TS–Int

Le Tableau 4.2 rapporte les résultats détaillés de TS–Div pour plusieurs instances difficiles ($G$, $k$) – particulièrement celles qui n'ont pas été résolues par la recherche Tabou de base. Les Colonnes 1–3 décrivent les instances : le graphe G, le nombre chromatique $\chi$, la meilleure borne supérieure connue ($k^*$) du $G$, et le nombre de couleurs $k$ utilisé par TS–Div. Les colonnes 4–6 présentent les résultats de l'algorithme, i.e. la Colonne 4 indique le taux de succès (le nombre d'exécutions qui résolvent le problème en 50 heures *ou moins*) ; les Colonnes 5 et 6 représentent le nombre moyen d'itérations et, respectivement, le temps moyen requis pour trouver une solution. TS–Div améliore la recherche Tabou de base pour la plupart des instances difficiles parce que TS–Div n'arrête jamais d'explorer de nouvelles régions – voir également la Section 2.5.4 pour les résultats d'une recherche Tabou de base très similaire ($f$-RCTS emploie quelques éléments en plus, comme le critère d'aspiration).

| Instance | | | Algorithme TS–Div | | |
|---|---|---|---|---|---|
| $G$ | $\chi, k^*$ | $k$ | $\frac{\text{réussites}}{\text{exécutions}}$ | itérations [$\times 10^6$] | temps [heures] |
| $dsjc500.1$ | ?,12 | 12 | 10/10 | 42 | $< 1$ |
| $dsjc500.5$ | ?,48 | 48 | 2/10 | 7409 | 35 |
| $dsjc500.9$ | ?,126 | 126 | 10/10 | 473 | 2 |
| $dsjc1000.1$ | ?,20 | 20 | 2/10 | 2200 | 9 |
| $dsjc1000.5$ | ?,83 | 87 | 5/10 | 2464 | 28 |
| $dsjc1000.9$ | ?,224 | 224 | 8/10 | 1630 | 24 |
| $flat300\_28\_0$ | 28,28 | 29 | 7/10 | 1186 | 8 |
| $flat1000\_76\_0$ | 76,82 | 86 | 3/10 | 3020 | 33 |
| $le450\_25c$ | 25,25 | 25 | 4/10 | 765 | 11 |
| $le450\_25d$ | 25,25 | 25 | 2/10 | 1180 | 19 |
| $r1000.1c$ | ?,98 | 98 | 10/10 | 47 | $< 1$ |

TABLE 4.2 – Les résultats de TS–Div avec une limite de temps de 50 heures. Pour chaque instance de $k$-coloration (indiquée par Colonnes 1–3), la Colonne 4 rapporte le taux de succès, i.e. le nombre d'exécutions réussies sur 10 ; les Colonnes 5 et 6 indiquent le nombre moyen d'itérations et le temps moyen nécessaire pour réussir (atteindre une solution).

Le Tableau 4.3 présente les résultats de TS–Int sur plusieurs instances ; pour chaque instance, TS–Int explore la proximité d'une configuration d'entrée fournie par TS–Div. Les Colonnes 1–3 indiquent l'instance (comme les Colonnes 1–3 dans le Tableau 4.2), la Colonne 4 montre l'amplitude de l'amélioration (le nombre de conflits de la configuration de départ et de la configuration de *fin* – la meilleure trouvée par TS–Int), la Colonne 5 présente le taux de succès pour arriver à cette amélioration et les Colonnes 6–7 indiquent l'effort moyen de calcul (en itérations et en heures, respectivement).

| Instance | | | Amélioration | Taux de succès | Itérations [$\times 10^6$] | Temps [heures] |
|---|---|---|---|---|---|---|
| $G$ | $\chi, k^*$ | $k$ | $f_{\text{départ}} \longrightarrow f_{\text{fin}}$ | $\frac{\text{réussites}}{\text{exécutions}}$ | | |
| $dsjc1000.1$ | $?, 20$ | 20 | $1 \rightarrow 0$ | 10/10 | 3774 | 12 |
| $dsjc1000.5$ | $?, 83$ | 86 | $2 \rightarrow 0$ | 10/10 | 623 | 19 |
| | | 85 | $80^{*(k+1)} \rightarrow 0$ | 2/10 | 1453 | 39 |
| $dsjc1000.9$ | $?, 224$ | 223 | $1 \rightarrow 0$ | 10/10 | 23 | 4 |
| $flat300.28$ | $28, 28$ | 28 | $150^{*(k+2)} \rightarrow 0$ | 10/10 | $< 1$ | $< 1$ |
| $flat1000.76$ | $76, 83$ | 85 | $74^{*(k+1)} \rightarrow 0$ | 10/10 | 1655 | 36 |
| $le450.25c$ | $25, 25$ | 25 | $1 \rightarrow 0$ | 10/10 | 3410 | 10 |
| $le450.25d$ | $25, 25$ | 25 | $1 \rightarrow 0$ | 10/10 | 6466 | 25 |

TABLE 4.3 – Instances pour lesquelles TS–Int améliore une coloration d'entrée dans une limite de temps de 50 heures. Les colorations d'entrée sont fournies par TS–Div ; les cellules marquées * indiquent que TS–Int trouve une solution à $k$ couleurs uniquement à partir d'une solution à $(k+1)$ ou $(k+2)$ couleurs.

TS–Int peut atteindre une solution avec un taux de succès de 100% (voir la Colonne 5 du Tableau 4.3) s'il est lancé à partir d'une coloration adéquate (i.e. pas trop loin d'une solution, voir également la Section 4.4.3). De cette manière, TS–Int trouve pour la première fois une coloration légale avec 223 couleurs pour le graphe $dsjc1000.9$ qui a été bien étudié dans la littérature – notons que, très récemment, [Lü and Hao, 2010] vient de rapporter aussi une solution. Tandis que TS–Div assure la diversification, TS–Int est un algorithme d'intensification qui peut être systématiquement exécuté après TS–Div pour explorer presque exhaustivement un périmètre limité autour des meilleures configurations trouvées par TS–Div.

Il est important de noter que certaines configurations d'entrée fournies au TS–Int (dans le Tableau 4.3), peuvent être très facilement accessibles : sauf $dsjc1000.5$, $dsjc1000.9$ (et les cellules marquées *), toutes les configurations d'entrée ont été trouvées en une seule exécution TS–Div. Par exemple, nous avons résolu $(dsjc1000.1, k = 20)$ pour la première fois avec TS–Int, et non-pas avec TS–Div. La première exécution de TS–Div n'a pas trouvé de solutions, mais elle a trouvé trois colorations complètement différentes avec 1 conflit ; l'une d'entre elles peut systématiquement conduire TS–Int à la solution (taux de succès 100%). Des conclusions similaires peuvent être tirées pour les graphes de Leighton. Pour ces graphes, il est possible de trouver systématiquement une solution en appliquant TS–Int sur une première coloration avec un conflit trouvée par TS–Div (voir ci-dessous).

### 4.4.3 TS–Int – localisation exacte d'une solution à partir d'une localisation approximative

TS–Int peut être couplé avec tout algorithme qui peut fournir (ou "conjecturer") une localisation approximative de la solution. Dans notre procédé d'expérimentation, nous considérons que les configurations avec moins de conflits ont plus de chances d'être plus proches d'une solution ; ainsi, les meilleures configurations (selon $f$) trouvées par TS–Div sont fournies à TS–Int. Cependant, il y a d'autres possibilités pour "conjecturer" la localisation d'une solution ; notamment, il est raisonnable de supposer qu'une solution à $k$ couleurs pourrait être proche d'une solution à $(k+1)$-couleurs (ou même $(k+2)$ couleurs).

Cette hypothèse a fonctionné parfaitement pour le graphe $flat300.28$ pour lequel TS–Int trouve une solution à $\chi = 28$ couleurs à partir d'une coloration légale avec $\chi + 2$ couleurs (en fait, la $(\chi + 2)$-coloration a été d'abord transformée en une $\chi$-coloration en remplaçant les couleurs $\chi + 1$ et $\chi + 2$ par la couleur 1). Les graphes de cette famille sont construits en ajoutant des arêtes entre $\chi$ ensembles indépendants d'une $\chi$-partition initiale de $V$ [Culberson and Luo, 1996]. Une grande partie des 30 classes de la 30-coloration légale sont très proches d'une partie des 28 ensembles indépendants initiaux, et ainsi, TS–Int peut facilement reconstruire le reste de la solution. En effet, après avoir trouvé la solution à 28 couleurs, nous avons observé qu'elle était située à une distance de seulement $7\%|V|$ de la solution à 30 couleurs.

Nous avons expérimentalement observé que, si l'on fournit un point de départ situé à moins de $\frac{1}{4}|V|$ distance d'une solution, alors TS–Int trouve la solution avec un taux de succès de 100%. Nous avons testé ces observations sur plusieurs graphes et avec plusieurs colorations initiales, fournies à moins de $\frac{1}{4}|V|$ distance d'une solution connue. Pour rechercher une solution jusqu'à une distance de $\frac{1}{4}|V| = 25\%|V|$ autour de la configuration de départ, TS–Int a besoin de faire une exploration complète d'un arbre à 3 niveaux – chaque arête correspond à une distance de $10\%|V|$ dans l'espace de recherche, voir la Figure 4.2. Le nombre de niveaux qui peuvent être traités dans un certain délai dépend de la vitesse d'exploration et du degré moyen de l'arbre. Comme TS–Int pourrait facilement être parallélisé, il serait possible d'accélérer l'algorithme par un ordre de grandeur en lançant tous les processus de $C_s$ en parallèle. Le rayon de $\frac{1}{4}|V|$ pourrait ainsi être augmenté jusqu'à $40\%|V|$ et même jusqu'à $\frac{|V|}{2}$ (si plus de temps de calcul est accordé).

### 4.4.4 La structuration des graphes et du paysage de recherche

Nous avons observé que, bien que les colorations légales soient toujours très rares pour une instance difficile, le nombre des colorations à 1 conflit (atteintes par TS–Div) peut varier considérablement. Pour les graphes aléatoires, avant de trouver une première coloration légale, TS–Div visite normalement entre 3 et 20 colorations à 1 conflit. D'autre part, pour le graphe $flat300.28$, TS–Div peut descendre directement à une coloration légale, sans visiter de colorations à 1 conflit. À partir d'une coloration à 4–5 conflits, TS–Div fait 1–2 mouvements qui conduisent tout de suite à la solution. À l'autre extrême, pour les graphes de Leighton, TS–Div peut trouver des milliers de colorations à 1 conflit avant d'arriver à une première coloration sans conflit. C'est une raison pour laquelle, il

semble plus facile de résoudre l'instance $le450.25c$ en localisant d'abord une coloration à 1 conflit et en appliquant TS–Int sur celle-ci.

#### 4.4.4.1 Paysage de recherche pour les graphes de Leighton

L'explication de ces contrastes importants se trouve dans la structure du graphe à colorier : cette structure définit les contraintes entre les variables et cela définit le *paysage de recherche* – associé à *notre* voisinage et à notre fonction objectif. Les graphes de Leighton ont par construction une clique de $\chi = 25$ sommets [Leighton, 1979]. De nombreuses colorations à 1 conflit peuvent être similaires sur cette clique où il y a un dernier conflit difficile à réduire. À l'extérieur de cette clique, il y a de nombreuses possibilités pour colorier les autres sommets sans ajouter de conflits. Ainsi, ces nombreuses colorations forment des plateaux immenses sur le paysage de recherche, sur lesquels il est difficile de localiser une configuration légale. C'est pour cela que les distances entre les clusters des colorations à 1 conflit visitées en série par la recherche Tabou pour ce graphe (voir le graphique en bas à gauche du Figure 3.3, p. 51) sont plus petites que dans le cas d'un graphe aléatoire.

#### 4.4.4.2 Paysage de recherche pour les graphes $flat$

Les graphes $flat$ sont construits à partir d'une $\chi$-partition de $V$ : toute coloration légale affecte une couleur différente à chaque ensemble de cette $\chi$-partition. Si un algorithme parvient à trouver une coloration à 3–4 conflits, cette coloration devrait être très proche de la solution, car elle colore correctement la plupart de ces $\chi$ ensembles. En effet, la descente vers la solution a toujours été très rapide avec tout algorithme utilisant *notre* voisinage de base (i.e. fondé sur des changements de couleur, voir Section 2.2) ; il est possible de parvenir à une solution sans jamais passer par une coloration à 1 conflit. On peut dire que le paysage de recherche contient un seul puits profond presque vertical avec la solution au fond. Toute configuration avec peu de conflits se trouve autour de ce puits.

Cette structure particulière du paysage de recherche peut expliquer le comportement de plusieurs autres algorithmes sur une instance comme ($flat300.28$, $k = 28$). TS–Int peut localiser le puits simplement car il part d'une 30-coloration légale qui identifie implicitement la plupart des 28 ensembles indépendants initiaux. Cette coloration est située dans la proximité du puits et TS–Int ne peut pas "louper" le puits – toute cette proximité est explorée presque systématiquement. PartialCol [Blöchliger and Zufferey, 2008] peut très facilement trouver des solutions avec $k = 28$ couleurs simplement parce qu'il emploie une représentation et un voisinage à base de partitions et il exploite les mêmes particularités. VSS [Hertz *et al.*, 2008] peut résoudre l'instance avec $k = 28$ parce qu'un de ses espaces de recherche est fondé sur des partitions. Même les meilleurs algorithmes évolutionnistes et hybrides échouent sur ($flat300.28$, $k = 28$) parce qu'ils n'emploient pas une recherche locale à base de partitions. Pour l'instance plus facile ($flat300.28$, $k = 29$), nous avons observé que Evo–Div (l'algorithme du Chapitre 5) ne finit jamais en rapportant des colorations à 3–4 conflits : soit il trouve la solution, soit il ne trouve que des colorations à environ 10 conflits ou plus. Les algorithmes génétiques fonctionnent mieux sur les graphes

aléatoires où ils combinent des configurations complètement différentes pour atteindre de nouvelles régions. [2]

### 4.4.5 Temps d'exécution de TS–Div et de TS–Int

Pour TS–Div et TS–Int, la limite du temps de calcul a été fixée à 50 heures par exécution – pour toute instance. Nous avons vu que dans cette limite de temps, TS–Div et TS–Int peuvent atteindre de très bons résultats pour l'ensemble de graphes difficiles. D'ailleurs, de nombreuses solutions ont été trouvées en un temps bien plus faible.

Mentionnons que dans la littérature sur la coloration de graphe, il est courant de faire tourner un algorithme de coloration de plusieurs heures à plusieurs jours pour tenter de résoudre une instance difficile. Par exemple, une partie des recherches locales les plus récentes [Blöchliger and Zufferey, 2008; Hertz *et al.*, 2008] utilisent des limites de temps de 10 heures pour les plus grandes instances. De plus, pour le nombre d'itérations, les valeurs exigées par TS–Div sont très similaires aux autres algorithmes. En effet, le nombre maximum d'itérations exécutées par TS–Div est de l'ordre du milliard (entre $10^9$ et $8\times10^9$), dans le même ordre de grandeur avec d'autres valeurs rapportées dans la littérature – e.g. $2 \times 10^9$ dans [Blöchliger and Zufferey, 2008, Tableau 6] ou $3 \times 10^9$ dans [Hertz *et al.*, 2008, Tableau 5].

Au delà de ces considérations, un point plus important concerne la capacité d'un algorithme à trouver de meilleures configurations avec plus de temps de calcul. On comprend que, par sa nature, TS–Div trouvera continuellement de nouvelles sphères s'il y a plus de temps de calcul – il ne fait pas d'explorations redondantes sur le long terme.

Cela est une caractéristique souhaitable qui est vérifiée par peu d'algorithmes existants. Très souvent, en faisant tourner un processus de recherche plus longtemps, il n'est pas du tout garanti que les résultats s'améliorent – parce que la recherche peut se bloquer dans des optima locaux ou se coincer en ré-explorant les mêmes régions dans une boucle. En fait, TS–Div fournit une solution simple et efficace à ce problème sensible : il est contraint de découvrir de nouvelles sphères en permanence. La même remarque s'applique à TS–Int. Avec plus de temps de calcul, il explore plus de sphères, et, par conséquent, de meilleurs colorations peuvent être explorées.

## 4.5 Vers des applications à d'autres problèmes d'optimisation combinatoire

Si nous examinons en détail les algorithmes guidés – voir Algorithme 4.1 (p. 58) et Algorithme 4.2 (p. 62) – on peut constater que TS–Div et TS–Int exploitent très peu de propriétés spécifiques au problème de coloration. En effet, les seules composantes exigées sont :

2. Concernant le graphe de $flat1000.76$, tous les algorithmes existants sont très loin d'une solution à $\chi = 76$ couleurs, et ainsi, ils se comportent plutôt comme sur des graphes aléatoires. Nous avons observé que les 82-colorations trouvées par Evo–Div sont très différentes – par comparaison à celles de ($flat300.28$, $k = 28$) qui sont très similaires.

- un espace de recherche (dans notre cas, il comprend toutes les colorations possibles) ;
- une fonction de voisinage (dans notre cas, défini par des changements de couleur) ;
- une fonction objectif (dans notre cas, le nombre de conflits $f$) ;
- et une distance de l'espace de recherche (dans notre cas, la distance de transfert entre partitions).

Essentiellement, les deux algorithmes sont construits comme des extensions de la recherche Tabou. La condition nécessaire pour pouvoir les mettre efficacement en application est de trouver une mesure de distance : (i) avec une complexité de calcul raisonnable par rapport à une itération Tabou et (ii) qui reflète la proximité selon le voisinage utilisé – la recherche Tabou ne doit pas avancer sur une longue distance en une seule itération. Plus formellement, la distance entre deux configurations doit indiquer le nombre minimum de transitions de voisinage nécessaires pour aller d'une configuration à l'autre. Comme pour tout algorithme générique, le succès de TS–Div ou TS–Int dépend aussi de quelques facteurs dépendants du problème, i.e. un choix adéquat du rayon $R$, ou une méthode pour bien définir les configurations de grande qualité.

Par conséquent, s'il est possible de trouver une telle mesure de distance dont le temps moyen de calcul n'est pas trop élevé par rapport à l'exécution d'une itération Tabou, les deux algorithmes peuvent être appliqués. TS–Int est moins sensible à la vitesse de calcul de distance, et ainsi, il peut plus facilement être appliqué à tout problème. Il pourrait être très utile lorsqu'il y a une méthode quoi que ce soit de "conjecturer" la localisation approximative de la solution. Cependant, la mesure de distance devrait toujours regrouper dans une $R$-sphère tous les optima locaux avec le même "backbone" – partageant une sous-structure commune importante. On peut trouver plusieurs exemples de distances qui peuvent être définies de cette manière, en employant quelques voisinages spécifiques :

- la distance de Hamming pour les problèmes avec représentation de type chaîne (tableau unidimensionnel) utilisant un voisinage 1-Flip (i.e. des problèmes de satisfaction de contraintes avec un voisinage qui change simplement la valeur d'une variable),
- le distance de Kendall Tau [Kendall, 1938] pour des problèmes avec une représentation basée sur des permutations, utilisant un voisinage fondé sur des transpositions adjacentes (i.e. le voyageur de commerce avec un voisinage qui inverse deux villes adjacentes),
- la distance de Levenshtein pour des problèmes avec représentation de chaîne et avec un voisinage défini en utilisant des opérations d'édition (i.e. supprimer, insérer ou remplacer une position du chaîne)

En ce qui concerne le temps élevé du parcours d'archive, il peut être réduit significativement si besoin avec au moins trois méthodes : (i) en faisant le parcours uniquement sur les configurations de qualité (dans les couches "profondes" de l'espace de recherche), (ii) en augmentant le rayon $R$ et (iii) en transformant l'archive dans une file qui enlève le plus ancien élément à chaque opération d'insertion. Dans le dernier cas, l'algorithme TS–Div deviendrait une Recherche Tabou Double, i.e. avec deux listes : (1) la liste Tabou traditionnelle contenant des configurations récemment visitées qui sont interdites, (2) une nouvelle liste de sphères récentes qui sont utilisées pour que TS–Div évite les sphères visitées dans "le passé récent". La portée de l'expression "passé récent" dépendrait de la taille de la file qui devrait être réglée selon le ralentissement induit par la composante

d'apprentissage.

## 4.6 Conclusions de chapitre

Nous avons développé une méthode rapide pour enregistrer le chemin d'exploration d'une recherche locale. L'analyse empirique du Chapitre 3 a montré que les minima locaux visités par la recherche Tabou sont structurés en sphères de diamètre $\frac{1}{10}|V|$. Ainsi, TS–Div enregistre le chemin d'exploration en enregistrant les sphères et il utilise un processus d'apprentissage additionnel pour décourager tout retour vers des sphères déjà explorées. L'algorithme TS–Div n'ajoute aucun paramètre pour l'utilisateur, car $B_f$ (le seuil pour définir les configurations de grande qualité) est automatiquement réglé.

L'objectif principal de TS–Div est la diversification globale : par rapport à l'algorithme Tabou de base, TS–Div ne risque pas de faire trop d'explorations redondantes à long terme. Ainsi, TS–Div est beaucoup plus efficace que l'algorithme Tabou de base et il peut faire concurrence aux meilleurs algorithmes de la littérature – parfois fondés sur des techniques hybrides assez complexes. La capacité de recherche de TS–Div est renforcée par TS–Int, qui est un algorithme orienté vers l'intensification. Il explore minutieusement un périmètre limite autour des configurations les plus prometteuses trouvées avec TS–Div. Nous avons montré que TS–Int peut systématiquement trouver une solution si la solution est située à moins d'une certaine distance de la configuration de départ. TS–Int organise l'espace de recherche comme un arbre de sphères et applique un algorithme classique de parcours en largeur pour investiguer méthodiquement toutes les sphères.

Pour conclure, la paire d'algorithmes TS–Div/TS–Int permet d'atteindre les deux objectifs principaux de tout algorithme de recherche : diversification *et* intensification. Le Tableau 4.4 présente une comparaison entre les meilleurs résultats obtenus dans ce chapitre et les meilleurs résultats de la littérature. Cinq articles de ce tableau sont très récents (année 2008 ou 2009), car nous avons pris en considération uniquement les meilleurs résultats pratiques – plus d'algorithmes se trouvent dans la Section 1.2.4 (p. 19). Les heuristiques à base de populations (les quatre dernières colonnes) sont traditionnellement les plus efficaces. En effet, sur plusieurs instances (e.g. $dsjc1000.5$), elles trouvent toutes de meilleurs résultats par rapport aux recherches locales. Cependant, TS–Div/TS–Int rivalise vraiment avec ces algorithmes et il trouve pour la première fois une nouvelle borne supérieure pour l'instance $dsjc1000.9$, étudiée depuis presque une vingtaine d'années [Johnson *et al.*, 1991].

| Graphe | $\chi, k^*$ | TS–Div/Int | VSS | PCol | AmaCol | DCNS | HEA | MMT | MCol |
|---|---|---|---|---|---|---|---|---|---|
| | | 2009 | 2008 | 2008 | 2008 | 1993 | 1999 | 2008 | 2010 |
| $dsjc500.1$ | $?, 12$ | *12* | 12 | 12 | 12 | 12 | – | 12 | 12 |
| $dsjc500.5$ | $?, 48$ | *48* | 48 | 48 | 48 | 49 | 48 | 48 | 48 |
| $dsjc500.9$ | $?, 126$ | *126* | 126 | 126 | 126 | 126 | – | 127 | 126 |
| $dsjc1000.1$ | $?, 20$ | *20* | 20 | 20 | 20 | 21 | 20 | 20 | 20 |
| $dsjc1000.5$ | $?, 83$ | 85 | 87 | 88 | 84 | 88 | 83 | 83 | 83 |
| $dsjc1000.9$ | $?, 224$ [223] | ***223*** | 224 | 225 | 224 | 226 | 224 | 225 | 223 |
| $r1000.1c$ | $?, 98$ | *98* | – | 98 | – | 98 | – | 98 | 98 |
| $flat300.28$ | $28, 32$ | *28* | 28 | 28 | 31 | 31 | 31 | 31 | 29 |
| $flat1000.76$ | $76, 82$ | 85 | 86 | 87 | 84 | 89 | 83 | 82 | 82 |
| $le450.25c$ | $25, 25$ | *25* | 26 | 25 | 26 | 25 | 26 | 25 | 25 |
| $le450.25d$ | $25, 25$ | *25* | 26 | 25 | 26 | 25 | 26 | 25 | 25 |

TABLE 4.4 – Performance de TS–Div/TS–Int par rapport aux meilleurs résultats sur les graphes difficiles les plus étudiés. Toutes les colorations rapportées par TS–Div/TS–Int sont disponibles en ligne : `www.info.univ-angers.fr/pub/porumbel/graphs/tsdivint/`. Les acronymes des algorithmes correspondent aux articles cités en Section 1.2.4 (p. 19).

On indique entre crochets les bornes publiées (en-ligne) au cours de la rédaction de ce manuscrit (après TS–Div/TS–Int [Porumbel *et al.*, 2010]), cf. Section 1.2.4.

# Chapitre 5

# Contrôle de Diversité et Croisement Informé dans L'Approche Évolutionniste

Nous présentons une approche hybride évolutionniste pour la coloration de graphe. Elle repose à la fois sur des idées génériques et des techniques spécialement conçues. Une attention particulière est portée aux mécanismes qui permettent de contrôler l'écart entre les individus à l'aide de la mesure de distance. Nous présentons une procédure générique d'écart qui décide des individus acceptables dans la population, des individus qui doivent être enlevés et de l'application des mutations. De plus, nous introduisons un croisement multi-parent qui analyse plusieurs caractéristiques très pertinentes pour identifier les meilleurs blocs constitutifs (classes de couleurs) utilisés dans la construction de l'enfant. L'algorithme résultant (Evo–Div) peut être généralement considéré comme bien informé, puisque que chaque composante est fondée sur informations les plus pertinentes, identifiées à la fois par des observations empiriques et aussi par des analyses théoriques. Evo–Div permet de trouver toutes les meilleures solutions de la littérature (sauf 2) et même d'atteindre des bornes non connues avant la thèse. Le contenu de ce chapitre a été partiellement publié dans un article présenté à Evocop 2009 ([Porumbel *et al.*, 2009a], sélectionné parmi les trois meilleurs articles), et aussi dans un article long en cours de soumission [Porumbel *et al.*, 2009c].

## Sommaire

**5.1 Introduction . . . . . 76**
**5.2 Présentation générale de l'algorithme évolutionniste . . . . . 77**
5.2.1 Terminologie du calcul évolutionniste . . . . . 77
5.2.2 L'algorithme mémétique . . . . . 78

**5.3 Croisement "bien informé" . . . . . 79**
5.3.1 Les meilleures caractéristiques héritables et le nombre de parents 80
5.3.2 Travaux connexes . . . . . 82
**5.4 Préservation et création continue de diversité . . . . . 82**
5.4.1 Distance entre les individus . . . . . 83
5.4.2 Rejet des enfants . . . . . 83
5.4.3 Stratégie de remplacement . . . . . 86
5.4.4 Idées similaires dans la littérature . . . . . 88
**5.5 Résultats et discussions . . . . . 90**
5.5.1 Conditions expérimentales . . . . . 90
5.5.2 Résultats avec un temps standard de 300 minutes . . . . . 91
5.5.3 Comparaison avec les meilleurs algorithmes . . . . . 94
5.5.4 Influence de la diversité, du croisement, et de la fonction d'évaluation 95
**5.6 Conclusions . . . . . 97**

## 5.1 Introduction

Comme nous l'avons souligné dans l'introduction de cette thèse (voir Section 1.2.1), les algorithmes évolutionnistes hybrides ont traditionnellement atteint de très bons résultats pratiques et ils font partie des approches les plus efficaces pour la coloration de graphe (voir aussi Section 1.2.4). Dans ce chapitre, nous nous concentrons sur deux facteurs cruciaux dans le calcul évolutionniste : la diversité de la population et l'opérateur de croisement.

En effet, une difficulté connue des algorithmes évolutionnistes concerne la diversité et le risque de convergence prématurée. Pour tenter de réduire ce risque, nous utilisons la mesure de distance dans l'espace de recherche et nous montrons comment l'employer pour assurer un *écart* adéquat entre les individus. Nous présentons une stratégie générique pour gérer la population afin d'imposer une distance minimum entre tout couple d'individus. La nouvelle stratégie permet d'éviter la convergence prématurée, tout en conservant la qualité de population. Cette procédure permet également de réduire un autre risque spécifique pour les petites populations : celui de ne pas couvrir de manière adéquate l'espace de recherche [Reeves, 1997].

Une autre question délicate dans les algorithmes évolutionnistes est de concevoir un opérateur de croisement pertinent pour le problème en question, i.e. qui préserve les bonnes caractéristiques des parents et perturbe les mauvais [Radcliffe, 1994]. Pour la coloration de graphe, il est très utile de considérer une coloration comme une partition [Dorne and Hao, 1998a; Galinier and Hao, 1999]. En effet, les meilleurs croisements pour la coloration choisissent certaines classes des deux parents et celles-ci sont considérées comme blocs constitutifs (gènes ou groupes [Falkenauer, 1998]) à transmettre à l'enfant. Une question essentielle consiste à bien choisir les *meilleures* classes à utiliser pour engendrer un nouvel individu. Pour cela, à chaque classe est affecté un score qui évalue sa qualité. Tandis que les approches classiques considèrent uniquement la taille des classes ou leur nombre de conflits, nous présentons une mesure plus informée pour évaluer et trier les classes des

parents et, ainsi, pour mieux sélectionner l'information qui est transmise à la génération suivante.

Nous cherchons également à identifier les plus importantes caractéristiques qui rendent un individu très adapté (ou de grande qualité). Cette information est très utile pour déterminer un nombre approprié de parents, ainsi que pour la fonction d'évaluation dans la recherche locale. En utilisant l'hypothèse de clusterisation (i.e. on considère que les individus les plus adaptés sont regroupés dans des sphères de rayon $R$), nous définissons une frontière entre "individus proches" et "individus distants". Un principe important de notre stratégie de diversité consiste à travailler uniquement avec des individus distants (de plus de $R$). Notre algorithme (Evo–Div) atteint la plupart des meilleures bornes supérieures de la littérature et trouve aussi des solutions non connues avant la thèse. Enfin, la contribution de chaque nouvelle composante introduite par Evo–Div est clairement évaluée.

Dans la prochaine section, nous rappelons le modèle général de l'algorithme mémétique, ainsi que les définitions de base. Le nouveau croisement informé est détaillé dans la Section 5.3. La méthode de contrôle de diversité/écart est discutée en Section 5.4. La Section 5.5 est consacrée à des résultats expérimentaux et nous terminons ce chapitre par des conclusions.

## 5.2 Présentation générale de l'algorithme évolutionniste

### 5.2.1 Terminologie du calcul évolutionniste

Afin de présenter notre approche, notons que le vocabulaire lié à la communauté évolutionniste est différent de celui de la recherche locale. Nous préférons suivre la terminologie des algorithmes évolutionnistes :

- **individu** Les individus représentent les configurations (les solutions candidates) utilisées dans le cadre de la recherche locale. Par conséquent, nous disons qu'un ensemble d'individus forme la ***population*** $POP = \{I_1, I_2, \dots I_{|POP|}\}$ ;
- **fonction d'adaptation** $f$ Dans la communauté évolutionniste, on dit souvent que les algorithmes génétiques cherchent un individu ayant la meilleure valeur de la fonction d'adaptation (*fitness function,* en anglais). Dans notre cas, la fonction d'adaptation est équivalente à la fonction objectif (i.e. le nombre de conflits, voir Définition 1.4, p. 16) ;
- **fonction d'évaluation informée** $\widetilde{f_{\text{eval}}}$ Notre algorithme hybride emploie aussi une recherche locale qui utilise une fonction d'évaluation plus informée que la fonction objectif. Dans la Section 2.3 nous avons introduit *deux* nouvelles fonctions d'évaluation $\widetilde{f}_1$ et $\widetilde{f}_2$ ; dans ce chapitre, nous allons utiliser $\widetilde{f_{\text{eval}}} = \widetilde{f}_1$.

Cette nouvelle fonction d'évaluation est employée *uniquement* dans la procédure de recherche locale de l'algorithme mémétique ; en dehors de cette recherche locale, nous n'utilisons que le nombre de conflits (la fonction objectif) pour évaluer des individus.

### 5.2.2 L'algorithme mémétique

Le schéma d'Evo–Div (voir Algorithme 5.1) utilise les idées de base des algorithmes similaires précédents [Fleurent and Ferland, 1996a; Galinier and Hao, 1999; Costa *et al.*, 1995; Malaguti *et al.*, 2008; Dorne and Hao, 1998a; Galinier *et al.*, 2008; Morgenstern, 1996; Lü and Hao, 2010] mais le modèle traditionnel est enrichi avec quelques éléments :

- des routines pour contrôler la diversité de la population ;
- de nouveaux mécanismes pour contrôler l'application de la mutation ;
- la possibilité d'utiliser $n \geq 2$ parents.

La plupart des algorithmes évolutionnistes de coloration sont *mémétiques*, car le croisement ne peut pas généralement produire tout seul des individus de qualité suffisante ; une amélioration locale est (presque) toujours nécessaire. En effet, il est difficile d'agréger (coller) parfaitement les classes de couleur héritées, et ainsi, le croisement des parents de qualité peut conduire à un enfant moins adapté. Le but de la recherche locale est d'améliorer cette agrégation de classes et également d'intensifier la recherche autour de la coloration construite par le croisement.

Le schéma proposé (voir Algorithme 5.1) permet aux enfants d'être rejetés s'ils ne respectent pas certains critères d'écart (routine `acceptEnfant`). La mutation est déclenchée uniquement lorsque la reproduction classique ne peut plus générer d'individus suffisamment espacés, et quand on atteint un seuil maximum d'échecs (i.e. `maxRejets`). Comme dans le processus naturel d'évolution, la mutation se produit très rarement. La *condition d'arrêt* consiste à trouver une coloration légale, ou à atteindre une limite de temps prédéfinie.

**Algorithme 5.1** : Schéma de Base de l'Algorithme Évolutionniste Evo–Div

**Entrée :** L'espace de solutions candidat $\Omega$
**Valeur de retour :** la meilleure solution trouvée

1. Initialiser (aléatoirement) la population $Pop = \{I_1, I_2, \ldots, I_{|Pop|}\}$
2. **While** *condition d'arrêt* **non** atteinte
   - **A.** *rejets*$= 0$
   - **B. Repeat**
     1. $\{I_1, I_2, \ldots, I_n\}$=RandomParents($Pop, n$) /* $n \geq 2$ */
     2. $O$ =Croisement($I_1, I_2, \ldots I_n$)
     3. **If** *rejets*$\geq$*maxRejets*
        a. $O$ =Mutation($O$)
     4. $O$ =rechercheLocale($O$,*maxIter*)
     5. *rejets*++
   - **Until** acceptEnfant($Pop, O$)
   - **C.** $I_R$ = indivRemplacé($Pop$)
   - **D.** $Pop = Pop \backslash \{I_R\} \cup \{O\}$

La sélection des parents (`RandomParents`) est tout à fait classique : $n$ individus différents sont choisis uniformément et aléatoirement. Ainsi, le principe "les plus adaptés survivent" s'applique uniquement à l'étape de remplacement qui devient essentielle. Pour

compléter la description de l'algorithme, le reste du chapitre présente les “boîtes noires” utilisées dans ce schéma, notamment l'opérateur de croisement, (routine `Croisement`, voir Section 5.3) et la stratégie d'écart (routines `acceptEnfant`, `indivRemplacé` et `Mutation`, voir Section 5.4). La recherche locale (`rechercheLocale`) est indépendante de l'algorithme évolutionniste et elle est décrite dans le paragraphe suivant.

#### 5.2.2.1 La procédure de recherche locale

La routine `rechercheLocale` est l'algorithme introduit au 2ème Chapitre. Plus exactement, il s'agit de la recherche Tabou avec la nouvelle fonction d'évaluation à base de degrés, et avec la durée Tabou réactive, mais sans critère d'aspiration. Il est important de mentionner que la routine `rechercheLocale` renvoie finalement une *configuration aléatoire* parmi toutes les configurations visitées qui ont le même nombre minimum de conflits.

Nous rappelons brièvement les plus importantes particularités de notre approche : (i) la durée Tabou et (ii) la nouvelle fonction d'évaluation. La durée Tabou est $T_\ell = \alpha * f(C) + \texttt{random}(0, A) + \left\lfloor \frac{M_{\text{cst}}}{M_{\max}} \right\rfloor$, où $\alpha$, $A$ et $M_{\max}$ sont des paramètres prédéfinis, $M_{\text{cst}}$ est le nombre des derniers mouvements consécutifs sans variation du nombre de conflits (voir Section 2.4). Concernant la fonction d'évaluation, nous avons précisé en Section 2.3.1 qu'il pourrait être très utile d'introduire des critères additionnels pour différencier les colorations avec le même nombre de conflits. Dans ce chapitre, nous allons utiliser la fonction d'évaluation $\widetilde{f}_1$ (2.5) de la Section 2.3.3.1, et, comme il n'y a pas de risque de confusion avec $\widetilde{f}_2$, on va noter $\widetilde{f_{\text{eval}}} = \widetilde{f}_1$. Formellement, $\widetilde{f_{\text{eval}}}(I) = \sum_{\{i,j\} \in CE(I)} \left(1 - \frac{1}{2|E|\delta_i} - \frac{1}{2|E|\delta_j}\right)$, où $\delta$ indique le degré (toujours non-nul car $G$ est supposé connexe) et $CE(I)$ est l'ensemble des arêtes en conflits (cf. Section 1.2.2). En utilisant cette fonction, l'algorithme choisit toujours des mouvements menant au meilleur (minimum) nombre de conflit, mais le choix aléatoire est fondé sur cette fonction : plus la valeur $\widetilde{f_{\text{eval}}}$ d'une coloration est petite, plus il y a de chances qu'elle soit choisie comme prochaine coloration.

Le principe de base de cette fonction est le suivant : il est plus difficile de réduire le conflit d'une arête si les deux extrémités sont de degré plus élevé [Morgenstern, 1996]. Si l'algorithme doit choisir entre deux colorations avec un conflit, il est préférable de prendre celle avec l'arête en conflit la plus isolée, i.e. avec les deux sommets en conflit de degré inférieur, impliquant moins de contraintes. Ce principe est également utilisé dans d'autres composantes d'Evo–Div, notamment pour la recombinaison qui utilise les degrés de sommets afin de mieux distinguer les classes de couleur. La Section 2.5.2 sur la recherche locale, ainsi que la Section 5.5.4, montrent que ce principe est utile pour certains graphes avec une très forte variation de degré.

## 5.3 Croisement “bien informé”

Une coloration peut être vue comme une partition de l'ensemble des sommets (Définition 1.3), et cela semble la meilleure interprétation pour concevoir un croisement efficace.

Ainsi, un opérateur de croisement doit choisir $k$ classes de couleur des parents et les assembler pour construire l'enfant. Afin de pouvoir générer des enfants de qualité, le but est d'hériter les $k$ meilleures classes de couleur, i.e. celles qui apportent la contribution la plus importante sur la qualité. Par exemple, si l'on peut choisir $k$ classes sans conflit (ensembles indépendants) qui couvrent $V$, elles peuvent être employées pour construire une coloration légale. Cela n'est habituellement *pas* possible, et nous devons aborder une question essentielle : *comment* choisir *les k meilleures* classes dans plusieurs parents ? Pour cette question, nous proposons d'affecter des *scores* aux classes en employant une procédure informée, fondée sur plusieurs critères de valeur. Notons que nous introduisons un cadre généralisé qui détermine un nombre approprié de parents pour chaque instance.

Le premier critère pour noter une classe est fondé sur la qualité et repose sur le principe qu'une classe avec moins d'arêtes en conflit est toujours préférable à une classe avec plus de conflits. Cependant, on rencontre souvent des classes sans conflit car Evo–Div converge assez vite vers des colorations "presque optimales". Un deuxième critère est donc nécessaire : si l'on doit choisir entre deux classes avec le même nombre de conflits, on préfère la classe la plus grande. Enfin, pour les instances où les tailles des classes sont assez homogènes un troisième critère est employé pour davantage de discrimination : la somme des degrés des sommets. L'idée de ce troisième critère est qu'un sommet de haut degré serait plus contraignant et ainsi plus difficile à colorier correctement (voir également Section 2.3.1) ; en conséquence, une classe (bien colorée) avec des sommets de plus haut degré a plus de valeur.

Formellement, l'opérateur de croisement – dorénavant appelé WIPX (Well Informed Partition Crossover) – est spécifié dans l'Algorithme 5.2. D'abord, il cherche (étape 2.A et 2.B) dans tous les parents la classe avec le meilleur (minimum) score. Après l'affectation de celle-ci à l'enfant en cours de construction (étape 2.C), WIPX choisit la meilleure classe suivante et il itère ce procédé. À chaque étape, tous les scores de classe sont calculés en ignorant les sommets qui ont déjà reçu une couleur dans l'enfant (voir l'étape 2.A.1). WIPX s'arête quand $k$ classes de couleurs sont affectées ; tout sommet restant non colorié reçoit la couleur $k$ (étape 3).

Un risque potentiel de ce croisement est d'hériter la plupart des classes d'un seul parent, notamment s'il y a un parent très adapté avec des classes qui éclipsent les autres. Cependant, la similarité entre l'enfant et les parents est implicitement vérifiée dans une seconde étape par la stratégie de contrôle de diversité qui rejette l'enfant s'il est trop similaire à n'importe quel individu existant de la population.

### 5.3.1 Les meilleures caractéristiques héritables et le nombre de parents

Un principe important dans la conception de croisements pour les problèmes de groupement est d'éviter le plus possible de perturber les meilleures caractéristiques (gènes de qualité) des parents [Radcliffe, 1994]. Notre premier but est d'identifier en quoi consiste une caractéristique de qualité. La qualité d'une coloration de grande valeur peut résider dans de grandes classes individuelles (e.g. des grands ensembles indépendants), mais aussi dans une bonne *agrégation* de classes (i.e. la manière dont elles sont assemblées). Le nombre $n$ de parents détermine la fragmentation de l'agrégation des classes dans chaque parent.

**Algorithme 5.2** : Croisement “bien informé” entre partitions

**Entrée :** parents $I_1, I_2, \ldots, I_n$
**Valeur de retour :** enfant $O$

**1.** $O$ =vide, /*aucun sommet affété dans l'enfant $O$*/
**2.** **For** *currentColor*= 1 **To** $k$
  A. **Foreach** parent $I_i \in \{I_1, I_2, \ldots, I_n\}$
    **Foreach** classe de couleur $I_i^j$ de $I_i$
      1. Enlever de $I_i^j$ tous sommets déjà affectés en $O$
      2. $conflicts = |\{(v_1, v_2) \in I_i^j \times I_i^j \ : \ (v_1, v_2) \in E\}|$
      3. $classSize = |I_i^j|$
      4. $degreeCls = \sum_{v \in I_i^j} \delta_v$ /*$\delta_v$=degré de $v$*/
      5. $score[I_i^j] = conflicts - \frac{1}{|V|}(classSize + \frac{degreeCls}{|E| \times |V|})$
  B. $(i^*, j^*) = \arg\min_{(i,j)} score[I_i^j]$ /*s'il y a plusieurs (i,j) de score*/
    /*minimum, en choisir un aléatoirement*/
  C. **Foreach** $v \in I_{i^*}^{j^*}$
    $O[v]$ =*currentColor* /*$v$ est affecté une couleur*/
**3.** **Foreach** sommet non-affecté $v \in O$
  $O[v] = k$

En moyenne, chaque parent donne $\frac{k}{n}$ classes, et ainsi, un plus grand $n$ détermine un plus petit nombre de classes sélectionnées dans chaque parent – une plus forte perturbation de l'agrégation existante dans chaque parent. Notre but revient à décider de ce qui est le plus important : préserver l'agrégation de classes ou maximiser la probabilité d'hériter des classes de qualité ?

Certaines instances ont des solutions avec de nombreuses classes de très petite taille (e.g. classes de 3–4 sommets pour $djsc1000.9$ avec $|V| = 1000$ et $k = 223$). Il n'est pas difficile de trouver un ensemble indépendant de cette taille, mais il est difficile d'agréger de petites classes d'une manière optimale (sans chevauchement excessif). Cette situation apparaît typiquement pour les graphes denses, pour lesquels on a besoin d'un grand nombre de couleurs $k$. Ainsi, la taille moyenne de classe (i.e. $\frac{|V|}{k}$) est très faible. Dans cet exemple, il vaut mieux employer deux parents afin de préserver l'agrégation des classes existantes dans les parents.

D'autres instances (e.g. des graphes “creux”) nécessitent un très petit nombre de couleurs, mais la taille moyenne des classes est très grande, e.g. 50 sommets pour le graphe $dsjc1000.1$ avec $|V| = 1000$ et $k = 20$. Pour une telle instance, la valeur d'une coloration de bonne qualité réside dans de grands ensembles indépendants, plutôt que dans leur agrégation. En utilisant 4 parents, on obtient $4k = 80$ classes $I_i^j$ d'entrée dans l'Algorithme 5.2. C'est nettement moins que $4k \cong 1000$ classes que l'on obtenait pour le graphe $dscj1000.9$ évoqué plus haut. Ainsi, la recombinaison ne devient pas trop chaotique en employant plus de parents pour $dsjc1000.1$. De cette façon, la règle générique est d'employer deux parents pour des instances avec de petites classes, trois pour des instances avec des classes

moyennes, ou quatre pour de très grandes classes. Plus précisément, notre règle est la suivante :

1. $n = 2$ si $\frac{|V|}{k} < 5$
2. $n = 4$ si $\frac{|V|}{k} > 15$
3. $n = 3$ sinon.

### 5.3.2 Travaux connexes

L'Algorithme 5.2 définit un cadre général pour concevoir des opérateurs de recombinaison pour des problèmes de coloration, ou, plus généralement, pour des problèmes de partitionnement/regroupement. En fait, en modifiant la fonction *score de classe* (voir étape 2.A.5 de l'Algorithme 5.2), il est possible de générer d'autres croisements. En principe, il est possible d'obtenir GPX (Greedy Partition Crossover [Galinier and Hao, 1999]) en utilisant $n = 2$ parents et une fonction de score qui note chaque classe avec sa taille – de plus, il faudrait faire attention à alterner les parents qui produisent les classes. Les trois premiers croisements de [Malaguti and Toth, 2008] peuvent également être reproduits de cette manière, ainsi que le croisement de [Lü and Hao, 2010].

La recombinaison de [Hamiez and Hao, 2001] utilise un cadre différent (la métaheuristique "scatter search") et elle construit l'enfant *uniquement* à partir *d'ensembles indépendants*. Une version similaire peut être obtenue en mettant *score* $= \infty$ pour une classe avec des conflits et *score* = *-classSize* autrement. L'idée d'utiliser uniquement des ensembles indépendants pour la construction de l'enfant est présente dans plusieurs articles (e.g. voir aussi [Galinier *et al.*, 2008]) et certains de nos tests expérimentaux (voir Section 5.5.4) montrent qu'elle peut avoir une influence positive sur quelques graphes.

Dans toutes ces approches, après l'insertion de $k$ classes de couleurs dans l'enfant, certains sommets problématiques restent non coloriés. Habituellement, une procédure aléatoire ou gloutonne est utilisée pour affecter une couleur à ces sommets problématiques. Toutefois, nous estimons que cette opération entraîne également un risque de perturber certaines classes de couleurs de bonne qualité en ajoutant des conflits avec des décisions assez superficielles. Nous préférons affecter à tous ces sommets problématiques la même couleur $k$ (étape 3). Ainsi, seule la dernière classe de couleurs est perturbée par de nouveaux conflits. La recherche d'une meilleure affectation de couleurs pour ces sommets problématiques est donc laissée à la recherche locale à l'étape suivante.

Notons qu'il existe aussi d'autres types d'opérateurs de croisement dans la littérature, fondés sur la représentation vectorielle de colorations [Fleurent and Ferland, 1996b; Malaguti and Toth, 2008], sur l'unification des paires de sous-classes sans conflit [Dorne and Hao, 1998a], sur la reproduction sexuelle en divisant le graphe en deux parties [Marino *et al.*, 1999], "distance-preserving crossovers" [Tagawa *et al.*, 1999], etc.

## 5.4 Préservation et création continue de diversité

La stratégie de diversification présentée dans cette section repose sur deux objectifs. Le premier consiste à maintenir un écart approprié entre les individus à tout moment

de l'exécution. Cela permet de s'assurer que les individus de la population ne peuvent pas tous converger vers le même point de l'espace de recherche. Ainsi, la convergence prématurée est évitée et la diversité préservée. Toutefois, cela ne peut pas garantir que l'algorithme est également capable de *créer* de la diversité, i.e. de découvrir en permanence de nouvelles régions de l'espace. Pour cette raison, notre deuxième objectif consiste à faire *évoluer* continuellement la distribution spatiale de la population au cours du temps, pour passer continuellement des anciennes régions aux nouvelles. De cette manière, même si la population de notre algorithme mémétique est petite, elle peut couvrir de nombreuses régions de l'espace de recherche au cours du temps.

Il existe de nombreux moyens différents pour mesurer la diversité, mais dans cette étude, nous employons un indicateur fondé sur les valeurs de distance entre les individus. Notre diversité est ainsi basée sur un indicateur d'écart $S$, défini comme la distance moyenne d'un individu à l'autre. L'*écart minimum* $S_{\min}$ représente la distance minimum entre deux individus de la population. Les deux objectifs de notre stratégie d'écart peuvent être exprimés comme suit :

– garder $S_{\min}$ au dessus d'un seuil d'écart minimum cible $R$
– rendre l'écart moyen $S$ aussi élevé que possible.

En principe, le premier point est abordé par la procédure de rejet de l'enfant et le deuxième par l'opérateur de remplacement. Ceux-ci correspondent aux routines `acceptEnfant` et `indivRemplacé` dans l'Algorithme 5.1. Ils sont présentés respectivement dans les Sections 5.4.2 et 5.4.3 ci-dessous.

### 5.4.1 Distance entre les individus

Nous rappelons brièvement le principe de la distance de transfert utilisée par la stratégie d'écart. Nous définissons la fonction de distance entre les individus $I_A$ et $I_B$ en utilisant le codage sous forme de partition (voir la Définition 1.3, Section 1.2.2). La fonction de distance de transfert $d$ est la suivante : le nombre minimum d'éléments qui doivent être transférés entre les classes de la première partition $I_A$ de sorte que $I_A$ devienne équivalente à $I_B$. Cette distance indique aussi le nombre minimum de transitions de voisinage (changements de couleur) que la recherche locale doit faire pour passer d'$I_A$ à $I_B$.

Les idées principales de la procédure de calcul de distance sont décrites en Section 4.2.2.1 ; les détails complets sont disponibles en Chapitre 6. Cependant, la rapidité du calcul n'est pas critique pour l'algorithme évolutionniste car nous n'avons observé aucun ralentissement significatif provoqué par les calculs des distances (l'opérateur de recherche locale consomme le plus de temps). Finalement, rappelons que la distance peut prendre uniquement des valeurs entre 0 et $|V|$, et ainsi, les valeurs des distances sont habituellement rapportées en termes de pourcentages de $|V|$.

### 5.4.2 Rejet des enfants

Puisque le premier objectif de la stratégie de diversité est de maintenir un écart minimum cible, nous insérons un nouveau individu dans la population seulement si sa distance à chaque individu existant est plus grande que $R$ (l'enfant ne devrait pas être dans une

$R$-sphère d'un individu existant). Par conséquent, si un enfant $O$ est situé à une distance inférieure à $R$ d'un individu existant $I$, la procédure de rejet détecte ce problème (la routine `acceptEnfant` retourne False en Algorithme 5.1) et, de plus, elle effectue une des actions suivantes :

1. rejeter $O$, **si** $f(O) > f(I)$ – i.e. si $O$ est plus mauvais que $I$ ;
2. remplacer directement $I$ avec $O$, **si** $f(O) \leq f(I)$.

Toutefois, dans les deux cas, nous considérons que $O$ n'est pas satisfaisant car il n'apporte pas de diversité à la population. Dans cette situation, Evo–Div ne passe pas à la génération suivante et reprend tout le processus de reproduction – voir la boucle `Repeat-Until` à l'étape 2.B. d'Algorithme 5.1. Ce processus de reproduction est répété jusqu'à ce que l'enfant résultant soit suffisamment distant de tous les individus existants, voir également la Section 5.4.2.2.

#### 5.4.2.1 Détermination de l'écart minimum cible

Une question délicate dans cette procédure est de déterminer une valeur appropriée de l'écart minimum cible – noté $R$. Rappelons que $\mathscr{S}_R(I)$ représente la sphère de rayon $R$ centrée en $I$, i.e. l'ensemble d'individus $I' \in \Omega$ tels que $d(I, I') \leq R$ – voir Definition 4.1. Si $I$ est un individu de qualité, une valeur pertinente de $R$ devrait impliquer que tous les autres individus de qualité de $\mathscr{S}_R(I)$ partagent avec $I$ des classes importantes de couleur. Ainsi, ces individus ne pourraient pas apporter de nouvelles informations (e.g. classes essentielles de couleurs) dans la population. Nous devons déterminer la valeur maximum de $R$ telle que tous les individus structurellement *indépendants* de $I$ soient situés en dehors de $\mathscr{S}_R(I)$.

Comme tous les individus de la population sont des minima locaux obtenus avec la recherche Tabou, nous utilisons la valeur de $R$ déterminée à partir de l'analyse de chemin d'exploration de la recherche Tabou. Rappelons (Chapitre 3) ce scénario classique : partant d'un minimum local initial $I_0$, la recherche Tabou visite une séquence de colorations de qualité $I_0$, $I_1$, $I_2, \ldots I_N$ (i.e. nous ne comptabilisons que les individus qui vérifient $f(I_i) \leq f(I_0)$, $\forall i \in [1..N]$). Après l'enregistrement de tous ces individus jusqu'à $N = 40000$, nous avons calculé la distance pour chaque paire ($I_i$, $I_j$) avec $1 \leq i, j \leq N$ et nous avons construit un histogramme pour examiner le nombre d'occurrences de chaque valeur de distance.

Cet histogramme a montré (voir Section 3.3.4) que la distribution de la valeur de distance est bimodale, avec de nombreuses valeurs très petites (i.e. environ $5\%|V|$) et avec beaucoup de valeurs élevées. Cela met en évidence une structuration des meilleurs individus en clusters : les valeurs plus petites correspondent à des distances intra-cluster et les valeurs élevées à des distances inter-cluster. Si on note un "diamètre de cluster" par $C_d$, on peut dire que $C_d$ varie de $7\%|V|$ à $10\ \%|V|$, selon le graphe. Pour déterminer une valeur appropriée de $R$, il suffit de noter que deux individus distants de plus de $10\%|V|$ (la valeur la plus élevée de $C_d$) ne sont pas dans le même groupe – parce que (idéalement), ils contiennent des classes essentielles très différentes. En conformité avec la Section 3.3.4, nous considérons que deux individus distants de moins de $R = 10\%|V|$ sont très proches.

#### 5.4.2.2 Dispersion réactive de la population avec des mutations

Dans le meilleur des cas, la diversité est assurée uniquement par le processus "naturel" de reproduction, i.e. à l'aide du croisement et de la recherche locale. En effet, si l'enfant résultant est rejeté à cause du manque de diversité, Evo–Div essaie d'abord le processus naturel de reproduction encore une fois, voir la boucle `Repeat-Until` de l'Algorithme 5.1. Si besoin, Evo–Div peut aussi introduire de la diversité avec des mutations "artificielles", i.e. en perturbant certains sommets avant la recherche locale. Plus rigoureusement, la mutation efface d'abord les couleurs de $R$ sommets (choisis aléatoirement) et elle les reaffecte une couleur avec une procédure gloutonne. Si l'enfant résultant après la recherche locale est toujours rejeté (voir la boucle `Repeat-Until`), un nouvel enfant est produit et l'intensité de la mutation est également augmentée à $2R$ sommets, $3R$ .... Si le nombre de sommets perturbés atteint $|V|$, la mutation revient à générer une coloration complètement nouvelle.

Rappelons (Algorithme 5.1) que la mutation est appliquée uniquement quand le processus naturel de reproduction a échoué $maxRejets$ fois. La question suivante est importante : combien de rejets autorise-t-on, avant de recourir aux mutations pour assurer le seuil d'écart minimum $R$ ? En conformité avec le principe "assurer la diversité sans sacrifier la qualité", la mutation doit être uniquement un outil de dernier recours. Pour cela, le paramètre $maxRejets$ (voir l'Algorithme 5.1) doit être affecté une valeur d'un ordre de grandeur plus élevée que le nombre moyen d'enfants rejetés par génération. [1] Cependant, les mutations peuvent devenir plus fréquentes pour imposer de la diversité dans certaines situations spécifiques, en particulier quand le processus de recherche est bloqué sur des structures difficiles de l'espace de recherche.

**Hausse du seuil d'écart minimum $R$ pour disperser la population** La population peut converger vers un état stable dans lequel : (i) l'écart moyen est de moins de $2R$ et (ii) tous les individus ont la même valeur de la fonction d'adaptation (qui est aussi la meilleure trouvée). Cela indique qu'une grande partie de la population peut être confinée dans une sphère de rayon $2R$ qui contient de nombreux optima locaux distants de plus de $R$. Cette situation peut être associée à une structure particulière de l'espace de recherche : de nombreux plateaux confinés dans un puits profond. Nos techniques d'écart ne peuvent plus diriger le processus de recherche en dehors de cette sphère, et ainsi, un mécanisme de dispersion est proposé.

L'objectif de ce mécanisme est de déclencher de nombreuses mutations à la suite, et d'aider une partie de la population à quitter cette $2R$-sphère problématique. À cette fin, on peut également recourir à un autre outil très utile pour maintenir l'équilibre diversification/intensification : le rayon de sphère $R$ – qui indique aussi notre écart minimum cible. En doublant sa valeur dans cette situation, la plupart des enfants générés dans la $2R$-sphère sont rejetés. De plus, en divisant $maxRejection$ par dix, on permet moins d'es-

1. Ce nombre moyen d'enfants rejetés par génération peut être facilement déterminé en pratique, car il est indiqué par $\frac{\text{croisements–générations}}{\text{générations}}$ dans le Tableau 5.2, i.e. il est entre 0 et 3 selon le graphe. On a remarqué qu'une valeur $maxRejets = 50 \gg 3$ assure un taux de mutation en dessous de 0.1% dans la plupart des cas (voir aussi Section 5.5.1.1).

sais de reproduction naturelle (croisement et recherche locale) ; ceci a comme conséquence des mutations plus fréquentes et cela conduit à des enfants en dehors de la $2R$-sphère.

La même stratégie de dispersion est employée dans une autre situation très délicate : quand il y a trop d'enfants rejetés pendant une très longue période. Cette situation peut indiquer que la population est distribuée autour de quelques optima locaux avec des bassins d'attraction très grands qui font que le processus de reproduction naturelle génère des enfants toujours dans les mêmes bassins d'attraction. En déclenchant la procédure de dispersion, les mutations sont appliquées plus rapidement et une partie de la population quitte rapidement les positions actuelles. En pratique, la condition "trop d'enfants rejetés pendant une longue période" se définit simplement si nous prenons en considération le fait que le nombre moyen d'enfants rejetés par génération est entre 0 et 3 (voir Section 5.5.1.1). Ainsi, s'il atteint 5 pour toute l'exécution, le mécanisme de dispersion est appliqué.

### 5.4.3 Stratégie de remplacement

À chaque génération, un individu existant de la population est choisi par la routine `indivRemplacé` dans l'Algorithme 5.1 pour libérer une place pour un enfant. Tandis que le processus de reproduction est important pour découvrir de nouvelles régions prometteuses, cet opérateur de remplacement est également très important car il sélectionne des régions qui sont abandonnées. La position relative de l'enfant n'est pas prise en compte dans cette décision. Rappelons-nous que la procédure de rejet de l'enfant vérifie que ce dernier est suffisamment éloigné de tous les individus ; l'enfant sert à guider le processus de recherche vers une région nouvelle. De cette façon, tout au long du processus, la distribution de la population se déplace de régions déjà visitées vers de nouvelles régions.

#### 5.4.3.1 Remplacement direct

Le remplacement ne se fait pas uniquement avec la routine `indivRemplacé`. Tant que le processus de reproduction ne génère pas un enfant suffisamment différent de tous les individus existants, nous considérons que la distribution de la population est stable et Evo–Div ne passe pas à la génération prochaine. Comme cela est détaillé dans la Section 5.4.2, s'il y a un individu $I$ proche de l'enfant $O$, alors $I$ peut être directement remplacé par $O$ si $f(O) \leq f(I)$. Avec ce remplacement, la distribution de la population n'évolue pas réellement vers de nouvelles régions, mais la recherche dans la $R$-sphère contenant $O$ et $I$ est intensifiée. Puisque cette $R$-sphère contient au moins deux individus de qualité, nous considérons qu'elle est prometteuse et qu'elle pourrait mériter plus d'intensification.

Ce *remplacement direct* peut également conduire à des violations des contraintes de l'écart minimum cible. Par exemple, considérons deux individus $I_1$ et $I_2$ tels que $d(I_1, I_2) > R$, $d(O, I_1) < R$, et $d(O, I_1) < R$. Le remplacement direct de $I_1$ conduirait à un écart minimum de $d(O, I_2) < R$. Cette anomalie est résolue au prochain appel de `indivRemplacé`. En effet, cette routine trouve d'abord les individus les plus proches $I_a$ et $I_b$, et, si $d(I_a, I_b) < R$, le moins adapté des deux est éliminé. De cette manière, sur les trois colorations initiales *proches* ($O, I_1$ et $I_2$), une seule survit finalement, et une place est libérée pour des individus explorant d'autres régions. Ainsi, la population ne stagne pas.

#### 5.4.3.2 Remplacement standard

Dans le cas standard où $d(I_a, I_b) > R \; \forall I_a, I_b \in Pop$, la routine `indivRemplacé` n'est pas censée assurer l'écart minimum cible. Son objectif est d'augmenter l'écart moyen. Pour cela, elle doit fortement décourager l'existence de petites distances entre les individus de la population. De plus, elle doit aussi faire respecter le principe "survie des meilleurs". Puisque le choix des parents est aléatoire, l'étape de remplacement est essentielle pour maintenir l'écart *ainsi que* pour assurer une qualité suffisante.

La procédure de remplacement standard (voir Algorithme 5.3 ci-dessous) sélectionne deux individus proches comme candidats pour l'élimination, et le moins adapté est éliminé. Pour choisir le premier candidat $C_1$, Evo–Div utilise une sélection pseudo-aléatoire guidée par certains critères de qualité (via la fonction `AcceptCandidat`). Le deuxième candidat $C_2$ est choisi en introduisant le *critère d'écart* suivant : $C_2$ est le plus proche individu de $C_1$ respectant les mêmes critères de qualité que $C_1$. [2]

**Algorithme 5.3** : La fonction de remplacement (élimination) d'Evo–Div

```
Entrée : population Pop = (I_1, I_2, ..., I_|Pop|)
Valeur de retour : l'individu à éliminer
1. Repeat
       C_1 = RandomIndividual(Pop)
   Until AcceptCandidat(C_1)                    /* acceptation liée à la qualité*/
2. minDist = |V|                          /*le nombre de sommets, la distance maximum*/
3. Foreach I ∈ Pop − {C_1}
       If d(I, C_1) <minDist
           If AcceptCandidat(I)
              • minDist = d(I, C_1)
              • C_2 = I
4. If f(C_1) < f(C_2)
       Return C_2
   Else
       Return C_1
```

La fonction `AcceptCandidat` sépare la première moitié de la population (les individus ayant une adaptation meilleure que la *médiane*), de la deuxième moitié de la population. Ainsi, cette fonction accepte toujours pour l'élimination un candidat $C_i$ de la deuxième moitié. Par contre, si $C_i$ appartient à la première moitié de la population, alors $C_i$ est accepté selon une certaine probabilité (en pratique, on a utilisé une probabilité de 0.5). De plus, le meilleur individu (i.e. *the best fit individual*) est protégé : il ne peut jamais être éliminé (principe d'élitisme), sauf si le nombre de meilleurs individus est trop grand (plus de la moitié de la population, en pratique). Dans ce cas très particulier, *tout individu* peut devenir un candidat à l'élimination.

Le rôle de la première moitié de la population est de maintenir en permanence un échantillon des meilleurs individus découverts tout le long de la recherche évolutionniste. La première moitié de la population reste assez stable par rapport à la deuxième moitié

2. En cas de choix multiple pour $C_2$, une sélection aléatoire est effectuée.

qui est une sous-population axée sur la diversité et qui change très rapidement. Cela pourrait rappeler les principes du "Scatter Search", une métaheuristique utilisant un ensemble d'intensification *et* un ensemble de diversification – voir [Hamiez and Hao, 2001] pour une implémentation pour la coloration de graphe.

### 5.4.4 Idées similaires dans la littérature

#### 5.4.4.1 Stratégies de diversification en optimisation combinatoire

Des stratégies permettant d'assurer la diversité peuvent être trouvés dans de nombreuses directions de recherche. Dans les algorithmes génétiques "diversity-guided" ou "diversity-controlling" [Ursem, 2002; Shimodaira, 1997; Zhu, 2003], on emploie un indicateur global de diversité afin de choisir les opérateurs génétiques et leur probabilité d'application. Par exemple, [Zhu, 2003] utilise un indicateur basé sur la distance moyenne de Hamming afin de régler les taux de mutation et de croisement. [Ursem, 2002] emploie des opérateurs d'intensification (sélection et recombinaison) quand un indicateur de diversité est élevé, ou des opérateurs de diversification (mutation) quand la diversité est faible. Dans cette piste de recherche, il n'y a pas vraiment besoin d'une mesure de distance entre les individus, mais seulement d'un indicateur global de diversité de la population. Par exemple, [Ursem, 2002] utilise un indicateur de dispersion statistique autour de la valeur moyenne sur chaque variable. Cela s'est révélé particulièrement efficace en optimisation continue. En général, la diversité de population peut être mesurée par de nombreux indicateurs, selon le champ d'application (voir les références dans [Zhu, 2003, p. 1]).

Une étude à base de distances est l'algorithme MA|PM [Sörensen and Sevaux, 2006], qui utilise aussi une procédure de rejet d'enfants. Dans MA|PM, si un enfant ne respecte pas les critères de diversité, il est immédiatement muté. Pour Evo–Div, l'approche est différente : nous préférons répéter le processus naturel de reproduction jusqu'à ce qu'il apporte une diversité "naturelle" à la population. Dans Evo–Div, la qualité n'étant pas sacrifiée au profit de la diversité, notre approche permet aussi de faire de la concurrence aux meilleurs algorithmes pour notre problème de coloration.

Nous mentionnons aussi que, dans les algorithmes mémétiques, il est classique d'accorder de l'attention au fait que l'enfant doit être suffisamment différent de ses parents. Puisqu'une recherche locale est employée, il y a un risque élevé que la combinaison de deux parents proches mène à des solutions semblables. Pour éviter cela, une idée pertinente consiste à toujours croiser des parents suffisamment éloignés. De plus, on peut également souhaiter générer l'enfant à égale distance de ses deux parents – voir [Galinier and Hao, 1999; Tagawa *et al.*, 1999] pour des applications de ce principe en coloration. Des techniques de ce type peuvent être trouvées dans de nombreux problèmes d'optimisation combinatoires, e.g. le problème de voyageur de commerce, voir le Distance Preserving Crossover [Freisleben and Merz, 1996]. Cependant, notre croisement n'est pas prévu pour assurer des distances égales entre l'enfant et ses parents.

La "distance crowding" introduite dans [Deb *et al.*, 2002] est utilisée pour trier les individus en optimisation combinatoire multi-objectif. Une différence essentielle est que cette "distance crowding" est mesurée dans l'espace de la fonction objectif, i.e. elle est

fondée sur les différences entre les valeurs de la fonction d'adaptation [Deb *et al.*, 2002; Coello *et al.*, 2004; Rajagopalan *et al.*, 2008].

#### 5.4.4.2 Formation de niches en optimisation continue multimodale

L'optimisation continue multimodale a pour objectif de trouver tous les optima globaux d'une fonction. Dans ce contexte, plusieurs méthodes de niche (e.g. "crowding" ou "fitness sharing" [Mahfoud, 1995a; Mahfoud, 1995b; Smith *et al.*, 1993; Deb and Goldberg, 1989; Miller and Shaw, 1996; Goldberg and Richardson, 1987; Beasley *et al.*, 1993; De Jong, 1975; Cedeño and Vemuri, 1999; Smith *et al.*, 1993]) ont été introduites pour la « formation et la préservation de sous-populations » stables [Mahfoud, 1995a; Mahfoud, 1995b]. Chaque sous-population est concernée par un pic de la fonction multimodale et peut être interprétée intuitivement comme une sous-espèce qui exploite une "niche écologique".

Par rapport à cette vision, en optimisation discrète il nous semble délicat d'encourager la population à se spécialiser sur des niches fixes, le processus de recherche devrait passer en permanence d'une région à une autre. Pour les fonctions multimodales, les niches correspondent aux quelques pics de la fonction objectif. De cette façon, le croisement à l'intérieur d'une seule sous-population est même encouragé [Miller and Shaw, 1996; Cedeño and Vemuri, 1999] (croisement intra-niche). Dans un algorithme mémétique, le croisement "intra-niche" est préfèrable, car il permet de découvrir de nouvelles régions. Notre but est de faire évoluer la distribution d'une petite population très rapidement afin d'explorer de nombreuses "niches" différentes au fil du temps, i.e. pour *créer* continuellement de la diversité tout au long de la recherche.

Le "Fitness sharing" [Goldberg and Richardson, 1987] est une technique de niche très populaire basée sur le principe suivant : si deux individus sont distants d'au moins $\sigma_{\text{share}}$ (le rayon de niche, ou "sharing parameter"), alors leurs valeurs d'adaptation sont mutuellement pénalisées [Goldberg and Richardson, 1987; Miller and Shaw, 1996; Smith *et al.*, 1993; Deb and Goldberg, 1989; Beasley *et al.*, 1993]. Bien que cette approche soit différente de notre stratégie, un point commun est la définition d'un rayon de niche approprié – similaire à notre écart minimum cible $R$. Si notre stratégie refuse l'insertion d'enfants sur un rayon de $R$ autour de tout individu, le "fitness sharing" ne fait que pénaliser les valeurs de la fonction d'adaptation pour des individus distants de moins de $R$. Dans les deux cas, si un rayon inapproprié est utilisé, l'efficacité est considérablement diminuée. Nous avons trouvé une bonne valeur de rayon en utilisant une analyse du groupement des meilleurs individus ; en "fitness sharing", des théories sur le rayon sont disponibles (voir [Beasley *et al.*, 1993, §5.1]) et elles s'appuient souvent sur des hypothèses sur la fonction continue multimodale (disant, par exemple, qu'il y a un nombre a-priori connu de pics uniformément répartis).

Une autre méthode de niche est le "crowding" (e.g. de Jong's crowding [De Jong, 1975], Mahfoud's deterministic crowding [Mahfoud, 1995a], voir aussi [Cedeño and Vemuri, 1999]) où l'on essaie de construire des sous-populations de niche « en forçant de nouveaux individus à remplacer ceux qui sont similaires génétiquement » [Smith *et al.*, 1993]. De cette manière, un nouvel individu remplace toujours un individu de sa propre sous-population et il minimise le changement sur la population existante, i.e. il préserve la diversité déjà existante. Cela peut rappeler notre remplacement direct. Mais, tandis que notre opération

induit également des niches, les niches stables ne sont pas recommandées dans un espace discret. Le remplacement standard d'Evo–Div choisit l'individu éliminé indépendamment de la position de l'enfant dans l'espace de recherche ; la distribution de la population encourage de niches en niches à évoluer en permanence .

Pour conclure, le contrôle explicite de population utilisant des mesures de distance semble avoir reçu relativement peu d'attention en optimisation discrète mono objectif. Cependant, cette idée peut être très utile pour améliorer les algorithmes mémétiques avec de petites populations. Cette approche est déjà considérée parmi les plus efficaces pour de nombreux problèmes d'optimisation combinatoire. Le calcul de $|Pop|^2$ distances par génération est une limitation dans les algorithmes de niche [Miller and Shaw, 1996; Smith *et al.*, 1993], mais cela ne pose pas de problèmes dans l'approche mémétique, où la recherche locale est la phase qui consomme le plus de temps. Dans un algorithme multimodal, les techniques de niche sont également utilisées pour faire de l'intensification (car on garde des sous-populations stables sur les niches), mais, dans l'algorithme mémétique, l'intensification peut être assurée efficacement par la recherche locale. Dans notre cas, la politique de diversification peut se concentrer uniquement sur le guidage de la recherche vers de nouvelles régions.

## 5.5 Résultats et discussions

### 5.5.1 Conditions expérimentales

L'algorithme Evo–Div a été testé sur toutes les 47 instances DIMACS présentées dans la Section 1.2.3. Pour les 27 instances faciles indiquées dans la Section 1.2.3.1, Evo–Div trouve toujours la meilleure solution connue en quelques secondes ou quelques minutes au maximum. De plus, parmi les autres 20 (i.e. $47 - 27$) instances difficiles, Evo–Div peut aussi colorier en quelques secondes les graphes $le450.15c$ et $le450.15d$ avec $k = \chi = 15$ couleurs ; ainsi, nous nous focalisons sur les 18 instances qui restent réellement difficiles pour Evo–Div.

#### 5.5.1.1 Paramètres

Le réglage des paramètres est une tâche relativement simple pour Evo–Div : une valeur appropriée peut être affectée à chaque paramètre à partir de quelques considérations théoriques assez génériques. En recherchant une valeur plus optimale pour chaque paramètre, on pourrait certainement encore améliorer Evo–Div, mais vraisemblablement insuffisamment pour bouleverser nos conclusions principales. Les valeurs des principaux paramètres sont résumés dans le Tableau 5.1 et nous présentons ci-dessous les motivations qui nous ont conduits vers ces valeurs.

– Paramètres génétiques majeurs : (i) la taille de population est $|POP|$=20, (ii) le nombre de parents $n$ est entre 2 et 4, et (iii) l'écart minimum cible est $R = 10\%|V|$. Notre stratégie d'écart est prévue pour une petite population, et ainsi, toute taille de population entre 10 et 30 produirait probablement des résultats similaires. Le nombre de parents est par défaut $n = 2$ dans les travaux précédents, mais nous avons montré

comment le changer automatiquement selon des informations spécifiques à chaque instance (i.e. la relation entre $k$ et $|V|$, voir la Section 5.3.1). Rappelons que l'écart minimum $R$ a été réglé à $10\%|V|$ en exploitant les résultats de l'analyse de l'espace de recherche (voir Section 5.4.2.1).

– Le nombre d'itérations de la recherche locale est $maxIter = 100.000$ et les paramètres internes sont : $\alpha = 0.6$, $A = 10$ et $M_{\max} = 1000$ d'après des travaux précédents sur la recherche Tabou (voir aussi Section 5.2.2.1).
– Paramètres pour des cas spécifiques : (i) $maxRejets$ – le nombre maximum d'enfants rejetés avant de recourir aux mutations, et (ii) l'intensité des mutations. Une valeur de $maxRejets = 50$ semble être suffisante pour préserver la diversité sans trop sacrifier la qualité – i.e. on utilise très rarement ces mutations qui peuvent détériorer la qualité. Le nombre moyen d'enfants rejetés par génération est indiqué $\frac{\text{croisements–générations}}{\text{générations}}$ dans le Tableau 5.2, i.e. il est normalement entre 0 et 3. Nous divisons $maxRejets$ par 10 dans la phase de dispersion, afin de laisser l'algorithme recourir aux mutations plus rapidement et diversifier plus facilement. L'intensité de la mutation est réglée afin de perturber $R$ sommets, ce qui est suffisant pour produire une coloration à l'extérieur de la $R$-sphère de la coloration initiale. Un autre principe consiste à utiliser une intensité graduellement croissante – i.e. si le premier enfant produit par mutation est rejeté, la prochaine mutation perturbe $2R$ sommets, puis $3R$, $4R$, ... (voir aussi Section 5.4.2.2).

| Paramètres | Section | Rôle | Valeur(s) |
|---|---|---|---|
| $\|Pop\|$ | Algo. 5.1 | Taille de la population | 20 |
| $n$ | §5.3.1 | Nr. de parents (paramètre auto-adaptatif) | 2–4 |
| $R$ | §5.4.2.1 | L'écart minimum | $10\%\|V\|$ |
| $maxIter$ | Algo. 5.1 | Nr. d'itérations dans la recherche locale | 100.000 |
| $maxRejets$ | §5.4.2.2 | Nr. de rejets avant de recourir aux mutations | 50 |

TABLE 5.1 – Résumé des paramètres génétiques.

### 5.5.2 Résultats avec un temps standard de 300 minutes

Le Tableau 5.2 présente les résultats standard d'Evo–Div sur les instances difficiles avec une limite de temps de 300 minutes (5 heures). Les colonnes 1 et 2 décrivent l'instance : le graphe et le nombre de couleurs $k$. Pour chaque instance, ce tableau rapporte le taux de réussite sur 10 exécutions indépendantes (Colonne 3), le nombre moyen de générations nécessaires pour trouver une solution (Colonne 4), le nombre moyen de croisements (Colonne 5) et le temps moyen en minutes (dernière colonne). Le nombre d'itérations de la recherche locale est en relation étroite avec le nombre de croisements parce que la recherche locale (avec $maxIter = 100000$) est appliquée une fois après chaque croisement. Nous avons également examiné Evo–Div sur les deux grands graphes $C2000.5$ et $C4000.5$, mais ceux-ci représentent un cas spécial nécessitant beaucoup plus de temps. Toutefois, Evo–Div peut trouver une solution pour ($C2000.5$, $K = 148$) en trois jours (taux de réussite 4/10), pour

| Graphe ($k^*$) | k | Succès/Essais | Générations | Croisements | Temps[min] |
|---|---|---|---|---|---|
| $dsjc$500.1 (12) | 12 | 10/10 | 301 | 428 | 1 |
| $dsjc$500.5 (48) | 48 | 10/10 | 370 | 373 | 7 |
| $dsjc$500.9 (126) | 126 | 8/10 | 1987 | 2157 | 63 |
| $dsjc$1000.1 (20) | 20 | 10/10 | 1658 | 2454 | 29 |
| $dsjc$1000.5 (83) | 83 | 9/10 | 2148 | 2439 | 136 |
| $dsjc$1000.9 (224) | 223 | 2/10 | 2872 | 3296 | 245 |
| $dsjr$500.1$c$ (85) | 85 | 9/10 | 562 | 4156 | 93 |
| $dsjr$500.5 (122) | 122 | 8/10 | 1028 | 2230 | 36 |
| $r$250.5 (65) | 65 | 9/10 | 3175 | 6423 | 48 |
| $r$1000.1$c$ (98) | 98 | 10/10 | 593 | 2240 | 98 |
| $r$1000.5 (234) | 238 | 9/10 | 953 | 1785 | 99 |
| $le$450.25$c$ (25) | 25 | 10/10 | 10029 | 14648 | 90 |
| $le$450.25$d$ (25) | 25 | 10/10 | 5316 | 7115 | 45 |
| $flat$300.28.0 (28) | 31 | 10/10 | 46 | 50 | 0 |
| $flat$1000.76.0 (82) | 82 | 10/10 | 1646 | 1884 | 110 |
| $latin\_square$(98) | 100 | 1/10 | 585 | 973 | 42 |

TABLE 5.2 – Résultats détaillés d'Evo–Div avec un temps (CPU) limite de 300 minutes sur toutes les instances difficiles. L'algorithme trouve la plupart des meilleures solutions avec un taux de succès élevé (Colonne 3) – la meilleure borne supérieure $k^*$ est indiquée entre parenthèses dans la Colonne 1.

($C$4000.5, $K = 271$) en trente jours et également pour ($C$4000.5, $K = 272$) en dix jours.

Bien qu'une limite de temps de 5 heures ne soit pas très élevée pour la coloration, Evo–Div trouve la plupart des meilleures solutions – voir également les autres algorithmes dans le Tableau 5.4. Si l'on considère l'ensemble de 47 graphes DIMACS de la Section 1.2.3 (i.e. y compris les instances faciles de la Section 1.2.3.1), Evo–Div atteint en 5 heures la meilleure borne connue pour 42 graphes sur 47, et trouve deux bornes inconnues avant cette thèse : ($dsjc$1000.9, $k = 223$) et ($C$4000.5, $k = 271$). Il n'atteint pas la meilleure borne connue uniquement pour 3 graphes sur 47. Notons qu'une de ces trois bornes, ($latin\_square$, $k = 98$), peut être atteinte avec une autre version d'Evo–Div (voir ci-dessous).

Nous avons utilisé une limite de temps comme condition d'arrêt parce que, dans notre cas, les indicateurs plus génériques (i.e. nombre d'itérations) sont moins significatifs et ils peuvent être facilement mal interprétés. Par exemple, une limite de générations fixée ne prendrait pas en considération le ralentissement causé par un nombre variable de rejets d'enfant et de calculs de distance. De plus, la complexité théorique d'une génération, d'un croisement, ou d'une itération peut être différente d'un algorithme à l'autre ; une comparaison de tels indicateurs pourrait être également biaisée. La plupart des algorithmes récents [Malaguti *et al.*, 2008; Blöchliger and Zufferey, 2008; Hertz *et al.*, 2008; Galinier

*et al.*, 2008] utilisent également un temps limite comme condition d'arrêt ; les meilleurs résultats sont habituellement atteints en moins de 10 heures. Les temps que nous avons rapportés sont obtenus sur un processeur Xeon à 2.8GHz, avec le langage C++ compilé avec l'option d'optimisation -O2 (gcc version 4.1.2 sous Linux).

#### 5.5.2.1 Vitesse et couverture exhaustive de l'espace de recherche

Même si nous avons considéré ci-dessus la limite de 300 minutes comme norme standard, nous présentons des résultats avec d'autres limites pour mieux évaluer Evo–Div. Le Tableau 5.3 donne les performances avec un temps limite de 30 minutes (Colonnes 2-6) et de 12 heures, respectivement (Colonnes 7-11). Les colonnes de ce tableau ont la même signification que dans le tableau précédent, i.e. l'interprétation des colonnes 2-6 (et 7-11, respectivement) est la même que pour les Colonnes 2-6 du Tableau 5.2.

| Graphe | Temps limite : 30 minutes | | | | | Temps limite : 12 heures | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | k | #succ | #gén | #cross | min | k | #succ | #gén | #cross | min |
| *dsjc*500.1 | 12 | 10/10 | 301 | 428 | 1 | 12 | 10/10 | 301 | 428 | 1 |
| *dsjc*500.5 | 48 | 10/10 | 370 | 373 | 7 | 48 | 10/10 | 370 | 373 | 7 |
| *dsjc*500.9 | 126 | 2/10 | 453 | 514 | 15 | 126 | 10/10 | 3741 | 4319 | 125 |
| *dsjc*1000.1 | 20 | 7/10 | 1466 | 1667 | 23 | 20 | 10/10 | 1658 | 2454 | 29 |
| *dsjc*1000.5 | 85 | 7/10 | 368 | 368 | 25 | 83 | 10/10 | 2943 | 3577 | 178 |
| *dsjc*1000.9 | 225 | 1/10 | 298 | 304 | 28 | 223 | 3/10 | 4559 | 5252 | 400 |
| *dsjr*500.1*c* | 85 | 1/10 | 107 | 739 | 16 | 85 | 10/10 | 792 | 5936 | 136 |
| *dsjr*500.5 | 122 | 5/10 | 422 | 593 | 10 | 122 | 9/10 [a] | 1659 | 4087 | 68 |
| *r*250.5 | 65 | 6/10 | 650 | 1411 | 10 | 65 | 10/10 | 3961 | 10124 | 78 |
| *r*1000.1*c* | 98 | 4/10 | 149 | 311 | 13 | 98 | 10/10 | 593 | 2240 | 97 |
| *r*1000.5 | 239 | 5/10 | 326 | 376 | 24 | 238 | 10/10 | 1661 | 2639 | 146 |
| *le*450.25*c* | 25 | 3/10 | 1660 | 1991 | 13 | 25 | 10/10 | 10029 | 14648 | 90 |
| *le*450.25*d* | 25 | 4/10 | 1593 | 1926 | 13 | 25 | 10/10 | 5316 | 7115 | 45 |
| *flat*300.28.0 | 31 | 10/10 | 46 | 50 | 0 | 31 | 10/10 | 46 | 50 | 0 |
| *flat*1000.76.0 | 83 | 1/10 | 401 | 402 | 29 | 82 | 10/10 | 1646 | 1884 | 110 |
| *latin_square* | 102 | 1/10 | 342 | 545 | 24 | 100 | 3/10 | 4189 | 6717 | 315 |

TABLE 5.3 – Les résultats d'Evo–Div avec deux limites de temps. En 30 minutes, Evo–Div trouve plusieurs solutions que d'autres algorithmes ne trouvent pas. Sur le long terme (12 heures), la politique de diversité assure un taux de réussite de 100% pour de nombreuses instances difficiles.

*a.* Dans ce cas, 15 heures ont été nécessaires pour atteindre un taux de réussite de 10/10.

Plusieurs conclusions intéressantes peuvent être tirées du Tableau 5.3. Evo–Div peut encore trouver de nombreuses solutions en 30 minutes ; cela est très rapide, car les temps

de calcul pour la coloration sont généralement donnés en heures ou même en jours. À notre connaissance, la plus petite limite de temps précédemment rapportée dans la littérature est de 60 minutes dans deux articles publiés en 2008 – voir les Tableaux 1-4 dans [Blöchliger and Zufferey, 2008] et les tableaux 1-3 dans [Hertz *et al.*, 2008]. Evo–Div trouve en 30 minutes de nombreuses solutions non atteintes par aucun algorithme de ces sept tableaux : ($DSJC1000.5$, $k = 85$), ($flat1000.76$, $k = 83$), ($le450.25c$, $k = 25$) et ($le450.25d$, $k = 25$). Cela montre d'abord l'efficacité de l'opérateur de croisement, car la stratégie de diversité est moins active à court terme.

L'efficacité de la stratégie de diversité est attestée par les résultats sur le long terme. En effet, les algorithmes d'optimisation combinatoire ont souvent des difficultés à améliorer la performance en poussant le temps limite au-delà d'un certain seuil. Cela est dû au fait qu'il est difficile de sortir d'un plateau ou d'arrêter de boucler sur un ensemble limité de plateaux. Notre stratégie de préservation et de *création* continue de la diversité est capable de surmonter ces difficultés. Les résultats sur le long terme donnent des preuves que Evo–Div peut assurer une large couverture de l'espace.

Ainsi, Evo–Div atteint un taux de réussite de 100% si il a suffisamment de temps de calcul. En règle générale, si Evo–Div peut résoudre une instance, il la résout *systématiquement*. Cette règle n'a que deux exceptions : l'instance ($DSJC1000.9$,223) qui est une instance qui demande plutôt une forte intensification (elle peut être résolue même sans stratégie de diversité, voir No-Div dans la Section 5.5.4), et aussi *latin_square* qui est l'un des rares points faibles d'Evo–Div.

### 5.5.3 Comparaison avec les meilleurs algorithmes

Le Tableau 5.4 montre une comparaison entre les résultats d'Evo–Div et l'état de l'art, i.e. les meilleurs résultats des algorithmes discutés dans la Section 1.2.4. Evo–Div trouve des solutions non rapportées dans la littérature avant la thèse, et, à l'exception de très rares points faibles, il atteint les meilleures bornes supérieures avec un fort taux de réussite. De plus, le temps nécessaire pour obtenir ces résultats est assez faible – voir l'en-tête du Tableau 5.4 (Lignes 4 et 5) qui donne le temps maximum requis par d'autres algorithmes, ainsi que leur année de publication – cela fournit des informations sur la génération de la machine utilisée.

En principe, nous présentons les résultats principaux d'Evo–Div en utilisant une seule version de l'algorithme et avec une configuration fixe des paramètres. Seules les graphes $C2000.5$ et $C4000.5$ ont nécessité plus de 300 minutes à Evo–Div, mais tous les algorithmes ont fait une exception pour ces graphes immenses qui nécessitent habituellement des jours ou semaines de calcul. Les trois meilleures bornes supérieures atteintes avec des versions différentes d'Evo–Div sont clairement indiquées. Notons que plusieurs colonnes individuelles du Tableau 5.4 rapportent en fait la meilleure performance de plusieurs algorithmes, ou, au moins, de plusieurs versions d'un même algorithme.

| Graph | $\chi/k^*$ | Evo–Div | | Recherches locales [a] | | | | Algorithmes mémétiques et hybrides [a] | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | k | (#hits) | ILS | VNS | PCol | VSS | DCNS | HGA | HEA | AMCol | MMT | MCol |
| | | 2009 | | 2002 | 2003 | 2008 | 2008 | 1996 | 1996 | 1999 | 2008 | 2008 | 2010 |
| | | 5 heures | | 1.6h | 3h | 10h | 10h | >24h | >24h | ≈3h | ≈3h | ≈10h | 5h |
| *dsjc*500.1 | ?/12 | 12 | (10/10) | 12 | – | 12 | 12 | – | – | – | 12 | 12 | 12h |
| *dsjc*500.5 | ?/48 | 48 | (10/10) | 49 | 49 | 48 | 48 | 49 | 49 | 48 | 48 | 48 | 48h |
| *dsjc*500.9 | ?/126 | 126 | ( 8/10) | 126 | – | 126 | 126 | – | – | – | 126 | 127 | 126 |
| *dsjc*1000.1 | ?/20 | 20 | (10/10) | – | – | 20 | 20 | – | – | 20 | 20 | 20 | 20 |
| *dsjc*1000.5 | ?/83 | 83 | ( 9/10) | 89 | 90 | 88 | 87 | 89 | 84 | 83 | 84 | 83 | 83 |
| *dsjc*1000.9 | ?/224 [223] | 223 | ( 2/10) | – | – | 225 | 224 | 226 | – | 224 | 224 | 225 | 223 |
| *dsjr*500.1*c* | 84/85 | 85 | ( 9/10) | – | – | 85 | 85 | 85 | 85 | – | 86 | 85 | 85 |
| *dsjr*500.5 | 122/122 | 122 | ( 8/10) | 124 | – | 126 | 125 | 123 | 130 | – | 125 | 122 | 122 |
| *r*250.5 | 65/65 | 65 | ( 9/10) | – | – | 66 | – | 65 | 69 | – | – | 65 | 65 |
| *r*1000.1*c* | 98/98 | 98 | (10/10) | – | – | 98 | – | 98 | 99 | – | – | – | 98 |
| *r*1000.5 | 234/234 | 238/237 [b] | ( 9/10) | – | – | 248 | – | 241 | 268 | – | – | 234 | 245 |
| *le*450.25*c* | 25/25 | 25 | (10/10) | 26 | – | 25 | 26 | 25 | 25 | 26 | 26 | 25 | 25 |
| *le*450.25*d* | 25/25 | 25 | (10/10) | 26 | – | 25 | 26 | 25 | 25 | – | 26 | 25 | 25 |
| *flat*300.28 | 28/28 | 31/29 [b] | (10/10) | 31 | 31 | 28 | 28 | 31 | 33 | 31 | 31 | 31 | 29 |
| *flat*1000.76 | 76/82 | 82 | (10/10) | – | 89 | 88 | 86 | 89 | 84 | 83 | 84 | 82 | 82 |
| *latin_square* | ?/98 | 100/98 [b] | ( 1/10) | 99 | – | – | – | 98 | 106 | – | 104 | 101 | 99 |
| *C*2000.5 | ?/150[148 [c]] | 148 | (4/10) | – | – | – | – | 150 | 153 | – | – | – | 148 |
| *C*4000.5 | ?/280[272 [c]] | 271 | (1/10) | – | – | – | – | – | 280 | – | – | – | 272 |

TABLE 5.4 – Les meilleures colorations trouvées par Evo–Div en moins de 5 heures et les résultats des meilleurs algorithmes. Les colorations d'Evo–Div sont publiquement disponibles à `http://www.info.univ-angers.fr/pub/porumbel/graphs/evodiv/`

*a.* Les acronymes des algorithmes sont disponibles dans la Section 1.2.4, p. 19.

*b.* Pour ces graphes, $k$ peut être réduit en utilisant des versions différentes d'Evo–Div : ($R$1000.5,k=237) a été résolu avec un croisement qui utilise que des classes sans conflits ; (flat300.28, $k = 29$) et (latin_square, $k = 98$) ont été résolus avec une recherche Tabou plus longue – voir Section 5.5.4.

*c.* Pour ces deux graphes très grands $C$2000.5 et $C$4000.5, nous avons utilisé une limite de temps de 3 et 30 jours, respectivement. Notons que les bornes supérieures entre crochets ont été rapportées très récemment (quand ce manuscrit était à moitié fini) dans [Lü and Hao, 2010].

### 5.5.4 Influence de la diversité, du croisement, et de la fonction d'évaluation

Le but de cette section est d'analyser l'influence des concepts les plus importants exploités par Evo–Div. L'algorithme pourrait obtenir de bons résultats même en excluant certaines composantes. Par exemple, si l'on utilise un croisement moins diversifiant, cette faiblesse peut être partiellement compensée par la stratégie d'écart. Pour cette raison, plusieurs instances ne sont pas très sensibles à de petits changements théoriques. Cependant, nous avons choisi dix instances représentatives, avec la plus importante variation de performance, et nous examinons plusieurs versions d'Evo–Div sur ces instances.

Le Tableau 5.5 montre le taux de réussite et le temps de résolution obtenus avec Evo–Div et avec cinq versions de l'algorithme obtenues en désactivant certaines composantes.

| Graphe | k | Evo–Div | | *No-Div* | | *Basic-Cross* | | *R-5%* | | *R-20%* | | No-$\widetilde{f_{\text{eval}}}$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | #hits | Min | #hits | Min | #hits | Min | #hits | Min | #hits | Min | #hits | Min |
| $DSJC1000.1$ | 20 | 10/10 | 29 | 0/10 | – | 4/10 | 211 | 9/10 | 37 | 10/10 | 31 | 10/10 | 26 |
| $DSJC1000.5$ | 83 | 9/10 | 136 | 0/10 | – | 6/10 | 246 | 8/10 | 80 | 10/10 | 132 | 9/10 | 99 |
| $DSJC1000.9$ | 223 | 2/10 | 245 | 1/10 | 110 | 2/10 | 220 | 2/10 | 183 | 0/10 | – | 0/10 | – |
| $DSJR500.1c$ | 85 | 9/10 | 93 | 0/10 | – | 10/10 | 55 | 0/10 | – | 10/10 | 21 | 3/10 | 76 |
| $DSJR500.5$ | 122 | 8/10 | 36 | 1/10 | 4 | 10/10 | 25 | 1/10 | 4 | 7/10 | 65 | 2/10 | 108 |
| $r1000.5$ | 238 | 9/10 | 99 | 1/10 | 19 | 1/10 | 250 | 2/10 | 27 | 6/10 | 76 | 0/10 | – |
| $flat1000.76.0$ | 82 | 10/10 | 110 | 2/10 | 90 | 7/10 | 236 | 8/10 | 99 | 7/10 | 159 | 7/10 | 84 |
| $le450.25c$ | 25 | 10/10 | 90 | 0/10 | – | 0/10 | – | 2/10 | 107 | 10/10 | 62 | 9/10 | 89 |
| $le450.25d$ | 25 | 10/10 | 45 | 1/10 | 42 | 0/10 | – | 1/10 | 16 | 10/10 | 86 | 10/10 | 87 |
| $latin\_square$ | 100 | 1/10 | 42 | 0/10 | – | 0/10 | – | 0/10 | – | 0/10 | – | 1/10 | 211 |

TABLE 5.5 – Comparaison entre la version standard d'Evo–Div et cinq versions obtenues en excluant certains composantes d'algorithme. Pour chaque version, nous fournissons le taux de succès (Colonnes "#hits") et le temps en minutes (Colonnes "Min"). Les moyennes sont calculées sur les exécutions réussies (la limite de temps est toujours de 5 heures).

Le temps limite est toujours de 300 minutes, le processeur et la machine sont identiques et le code source est le même – sauf la ligne pour l'activation/désactivation de la composante investiguée. Les résultats confirment les considérations théoriques sur les éléments correspondants aux versions Evo–Div suivantes :

1. *No-Div* (Colonnes 5 et 6) : Evo–Div sans stratégie de diversité – i.e. tout enfant est accepté et l'individu le plus mauvais est éliminé à l'étape de remplacement. En comparaison avec Evo–Div (Colonnes 3 et 4), *No-Div* est significativement plus mauvais. Bien qu'il puisse encore trouver quelques solutions, les taux de réussite stables d'Evo–Div sont perdus parce que la diversité n'existe plus ;
2. *Basic-Cross* (Colonnes 7 et 8 ) : Evo–Div avec un croisement de base, similaire avec GPX [Galinier and Hao, 1999] (voir Section 5.3.2). Bien que *Basic-Cross* puisse trouver 70% des solutions atteintes par Evo–Div, son plus grand problème est qu'il rend l'algorithme plus lent. Même en ignorant les 30% d'échec, *Basic-Cross* exige environ dix fois plus de temps de calcul (voir $DSJC1000.1$) et c'est pour cette raison qu'il obtient des taux de réussite inférieurs. Cependant, ce croisement reste encore efficace pour l'instance difficile ($DSJC1000.$, $k = 223$) ;
3. *R-5%* (Colonnes 9 et 10) : Evo–Div avec un écart minimum cible fixe $R' = 5\%|V|$. L'Evo–Div standard obtient des résultats systématiquement meilleurs que cette version. Cela confirme l'hypothèse de clusterisation qui recommande de garder une distance minimum de $R = 10\%|V|$ entre tout couple d'individus ;
4. *R-20%* (Colonnes 11 et 12) : Evo–Div avec un écart minimum cible fixe $R'' = 20\%|V|$. Cette version échoue sur deux instances et obtient des taux de réussite inférieurs

sur encore trois instances. Les résultats sont bons, parce que $R'' > R$ et, ainsi, cette version d'Evo–Div ne souffre pas d'un manque de diversité. En revanche, elle force une diversification excessive et ne réalise pas assez d'intensification. Ainsi, elle ne résout pas ($DSJC1000.9$, $k = 223$) qui a besoin d'une forte intensification – rappelons qu'elle était résolue même par *No-Div*. Cela confirme de nouveau nos considérations théoriques liées à l'écart minimum cible de $R = 10\%|V|$ ;

5. *No-*$\widetilde{f_{\text{eval}}}$ (Colonnes 13 et 14) : Evo–Div avec une recherche locale utilisant la fonction d'évaluation classique $f$ au lieux de la fonction $\widetilde{f_{\text{eval}}}$ à base de degrés (Section 5.2.2.1). L'effet positif de la nouvelle fonction $\widetilde{f_{\text{eval}}}$ est plus visible sur les graphes avec une grande variation de degré – en particulier les graphes géométriques ($r1000.5$, $DSJR500.5$ et $DSJR500.1c$), pour lesquels le degré maximum peut être d'un ordre de grandeur plus élevé que le degré minimum. Il est naturel que la différenciation de $\widetilde{f_{\text{eval}}}$ soit moins significative sur les graphes avec des degrés plus homogènes (les graphes de Leighton ou la plupart des graphes aléatoires). Notons que $\widetilde{f_{\text{eval}}}$ a une influence positive aussi sur ($DSJC1000.9$, $k = 223$), car ce graphe a des degrés très élevés. Toutefois, à l'exception de ce cas et des graphes géométriques, Evo–Div est toujours en mesure d'obtenir des résultats similaires en utilisant la fonction classique d'évaluation (le nombre de conflits).

Finalement, il est important de mentionner encore deux variantes simples d'Evo–Div qui peuvent améliorer les résultats sur certains graphes. Nous avons également examiné Evo–Div avec une recherche Tabou plus longue (i.e. $maxIter = 10.000.000$). En changeant seulement ce paramètre, Evo–Div résolue ($flat300.28$, $k = 30$) avec un taux de succès de 5/10 dans un délai de 5 heures, et il a même atteint une solution pour $flat300.28$ avec $k = 29$ (en 5 heures). Soulignons aussi que cette version a résolu ($latin\_square$, $k = 98$) en un délai de 7.5 heures (avec une limite de temps de 24 heures, le taux de succès devient 4/30). Ainsi, $latin\_square$ a été colorié avec $k = 98$ couleurs pour la première fois en presque 15 ans [Morgenstern, 1996]. De plus, un croisement fonctionnant exclusivement avec des ensembles indépendants conduit à des résultats améliorés pour quelques graphes géométriques particuliers. Utilisant ce croisement, Evo–Div a ramené le temps de résolution à quelques secondes pour ($dsjr500.5$, $k = 85$), et il a même réussi à colorier le graphe $r1000.5$ très difficile avec $k = 237$ couleurs.

## 5.6 Conclusions

Nous avons décrit un nouvel algorithme mémétique (Evo–Div) qui se distingue par l'introduction d'une stratégie d'écart qui permet à la fois de préserver *et de créer* continuellement de la diversité. De plus, le croisement multi-parent (WIPX) exploite un grand nombre d'informations pour mieux choisir les classes de couleurs utilisées dans la construction de l'enfant. Nous avons montré expérimentalement que ce croisement assure une convergence rapide vers des solutions (Section 5.5.2.1). D'autre part, la convergence prématurée est empêchée par la stratégie d'écart ; à l'aide de plusieurs mécanismes complémentaires de dispersion, Evo–Div assure une couverture très large de l'espace de recherche. Les résultats

expérimentaux montrent que la stratégie de l'écart peut conduire à un taux de succès de 100% en utilisant suffisamment de temps.

Les résultats globaux sur l'ensemble de 47 graphes DIMACS (voir Section 5.5.3) sont encourageants. Evo–Div trouve la meilleure solution pour presque toutes les instances, et il atteint également deux bornes supérieures non connues avant la thèse : (dsjc1000.9, $k = 223$) et (C4000.5, $k = 271$). Il pourrait être intéressant de noter que la coloration légale pour ($latin.square$, $k = 98$) a été (re-)trouvée pour la première fois en presque 15 ans [Morgenstern, 1996]. Evo–Div n'a que seulement deux bornes supérieures non-atteintes (pour les graphes $r1000.5$ et $flat300\_28$).

A notre connaissance, c'est la meilleure performance dans la littérature. Bien qu'il y ait plusieurs algorithmes très récents à base de populations (AmaCol [Galinier *et al.*, 2008], MMT [Malaguti *et al.*, 2008] et MemCol [Lü and Hao, 2010]), ils n'atteignent pas la meilleure borne connue pour *au moins 4 instances*. Nous mentionnons qu'un de ces algorithmes (MemCol) vient d'être publié très récemment et il obtient d'excellents résultats – y compris ($dsjc1000.9$, $k = 223$). Cependant, il ne trouve pas de meilleure borne par rapport à Evo–Div et il semble capable d'explorer presque 3 fois plus de configurations pour le même intervalle de temps de 5 heures (e.g. pour $djsc1000.9$, sa vitesse mesurée en "itérations par minute" est presque 3 fois plus grande).

La solution pour les deux bornes supérieures non-atteintes n'est pas reliée à la diversité de population, car ces bornes ont été atteintes uniquement par des méthodes très spécifiques. En effet, pour résoudre $flat300.28$ avec $k = 28$ couleurs on a besoin d'insister sur l'intensification [Porumbel *et al.*, 2010] ou d'utiliser un codage à base de colorations légales partielles [Blöchliger and Zufferey, 2008; Hertz *et al.*, 2008]. Quant à $r1000.5$, une solution à $k = 234$ couleurs a été trouvée uniquement en utilisant la méthode de la génération de colonnes [Malaguti *et al.*, 2008] ou d'autres techniques exactes [Prestwich, 2002].

Le dernier, mais non le moindre, avantage d'Evo–Div est le fait que la stratégie d'écart est suffisamment générale pour être aussi employée dans d'autres problèmes d'optimisation combinatoire. Le mécanisme de rejets des enfants est uniquement fondé sur le calcul des distances entre l'enfant et les individus existants ; la procédure de remplacement n'a pas besoin d'opérations spécifiques pour la coloration (voir Algorithme 5.3). Pour mettre en application notre algorithme mémétique avec garantie de diversité, il suffit de spécifier les trois composantes suivantes : un espace de recherche, une fonction d'adaptation et une mesure de distance entre les individus.

# Chapitre 6

# Distance de Transfert : Calcul Rapide et Interprétation Spécifique à la Coloration

Dans ce chapitre, nous présentons en détail la distance de transfert entre partitions, ainsi qu'un nouvel algorithme plus rapide de calcul. Une coloration est une partition de l'ensemble de sommets, et ainsi, cette distance a été souvent utilisée dans la coloration de graphe. Toutefois, la distance de transfert a une applicabilité bien plus générale. Elle est souvent utilisée dans des domaines très divers, associés avec des problèmes de classification ou regroupement – e.g. segmentation d'image ou clusterisation. Nous présentons notre nouvel algorithme *exact* pour calculer cette distance : nous donnons plusieurs conditions qui permettent de réduire d'un ordre de grandeur la complexité de la méthode communément utilisée à ce jour – i.e. réduction au problème d'affectation et application de l'algorithme hongrois. Le chapitre développe des idées d'un article soumis en 2008 [Porumbel *et al.*, 2009b].

**Sommaire**


**6.1 Introduction . . . . . 100**
6.1.1 Contexte et travaux connexes . . . . . 100
6.1.2 Pourquoi un nouvel algorithme ? . . . . . 101
**6.2 Définition formelle de la distance . . . . . 102**
**6.3 Calcul de la distance . . . . . 103**
6.3.1 Matrice de similarité $T$ en $O(|S|)$ . . . . . 104
6.3.2 Affectation maximale en $O(|S|)$ . . . . . 105
**6.4 Extensions . . . . . 108**
**6.5 Une étude de cas pour la coloration de graphe . . . . . 109**
**6.6 Remarques finales . . . . . 110**
6.6.1 Pseudocode de l'algorithme . . . . . 110

## 6.1 Introduction

Soit $k$ un entier positif et $S$ un ensemble.[1] Une $k$-partition de $S$ (équivalente à une $k$-coloration d'un ensemble $V$ de sommets) est un ensemble de $k$ classes (sous-ensembles, parties, ou cellules) deux à deux disjointes, qui forment un recouvrement de $S$. Plusieurs applications pratiques de partitionnement ou de clusterisation de données font fréquemment appel à des calculs de distance entre partitions. Des études précédentes ont montré qu'il est possible de faire ce calcul en temps polynomial – minimum $O(|S|+k^2)$ et maximum $O(|S| + k^3)$ – en utilisant une réduction directe au problème d'affectation.

Nous décrivons plusieurs conditions qui permettent à un nouvel algorithme de calculer la distance en $O(|S|)$ opérations. En pratique, ces conditions sont associées à des partitions relativement proches/similaires. Nous allons donner des arguments pour montrer que, sauf cas exceptionnel (i.e. partitions spécialement construites), toute distance inférieure à $\frac{|S|}{5}$ peut être calculée en $O(|S|)$. De plus, cet algorithme peut également être employé pour identifier un ensemble maximal de classes très similaires (entre deux partitions distantes, mais avec des classes communes).

### 6.1.1 Contexte et travaux connexes

Étant données deux $k$-partitions $P_1$ et $P_2$, la distance de transfert entre $P_1$ et $P_2$ est la suivante : le nombre minimal d'éléments qui doivent être transférés entre les classes de $P_1$ pour obtenir une partition égale à $P_2$. La similarité est définie comme suit : le nombre maximum d'éléments qui n'ont pas besoin d'être transférés pour pouvoir transformer $P_1$ en $P_2$. Il n'y a aucune restriction concernant les tailles des classes ; il peut exister des *classes vides*, ainsi qu'une classe égale à $S$.

A notre connaissance, cette définition de distance a déjà été énoncée il y a presque 50 ans – voir [Régnier, 1983 et 1965]. La méthodologie de calcul communément utilisée aujourd'hui a été présentée en 1981 [Day, 1981]. Le calcul revient à un problème d'affectation linéaire sur une matrice $k \times k$. Le problème d'affectation peut se résoudre par la méthode hongroise [Kühn, 1955] de complexité minorée par $O(k^2)$ et majorée par $O(k^3)$. Une description complète de cette méthodologie est disponible [Gusfield, 2002] et toutes les études récentes utilisent une approche similaire [Charon *et al.*, 2006; da Costa and Rao, 2004; Cardoso and Corte-Real, 2005; Konovalov *et al.*, 2005; Denœud and Guénoche, 2006; Talbi and Weinberg, 2007].

L'algorithme hongrois a été un résultat remarquable, pour avoir résolu dans un temps polynomial le problème d'affectation – qui est un problème de base en recherche opérationnelle. L'entrée de ce problème est normalement une matrice de coût $k \times k$, et ainsi, tout algorithme de résolution doit inévitablement exécuter au moins $O(k^2)$ pas pour lire cette matrice. De plus, la méthode hongroise exécute plusieurs opérations de complexité $O(k^2)$. Dans notre travail, on exploite le fait que l'entrée contient seulement $O(|S|)$ éléments (i.e. les deux partitions) ; notre algorithme fonctionne avec une matrice de coût creuse où seulement $|S|$ éléments (au maximum) sont utilisés. La meilleure approche pour

1. En concordance avec la terminologie de la théorie des ensembles, nous utilisons $S$ pour noter un ensemble quelconque ; nous notons $V$ un ensemble de sommets.

ce calcul ne devrait pas appliquer directement l'algorithme hongrois. Bien qu'il existe des algorithmes pour résoudre le problème d'affectation sur des matrices creuses [Carpaneto and Toth, 1983], la possibilité de réduire la complexité en dessous de $O(k^2)$ n'a pas été considérée.

D'un point de vue théorique, la distance de transfert satisfait plusieurs propriétés intéressantes. Par exemple, il est déjà connu qu'elle constitue une métrique dans l'espace des partitions [Cardoso and Corte-Real, 2005]. Des études plus avancées [Charon *et al.*, 2006] montrent que, bien que la distance s'étende de 0 à $|S|$, elle ne peut jamais atteindre certaines bornes supérieures (e.g. $|S| - \left\lceil \frac{|S|}{k} \right\rceil$). Une comparaison entre la fonction de distance et d'autres indices similaires (e.g. l'indice de Rand) est également disponible [Denœud and Guénoche, 2006] ; elle montre la distribution de plusieurs indices entre des partitions proches. La distribution des distances entre des partitions aléatoires est également bien étudiée, et permet de mieux interpréter les valeurs de distance [Denœud, 2008]. Il y a aussi des généralisations de cette mesure de distance dans la littérature [Berman *et al.*, 2007; Berger-Wolf *et al.*, 2007].

Une récente étude [Denœud, 2008] pourrait être particulièrement utile pour la coloration de graphe parce qu'elle montre empiriquement que la distance moyenne entre deux partitions aléatoires est autour de $69\%|S|$ (avec une petite déviation standard) pour $|S| = 100$ [Denœud, 2008, Table 1] [2]. Cela fournit des repères pour pouvoir interpréter les valeurs de distance entre colorations. Par exemple, une conséquence directe est que deux colorations situées à une distance de $50\%|V|$ ne peuvent pas être considérées comme complètement indépendantes. De plus, il est certain que le rayon $R = 10\%|S|$ utilisé pour définir nos sphères représente une valeur très petite.

### 6.1.2 Pourquoi un nouvel algorithme ?

L'algorithme proposé dans ce chapitre calcule la distance en $O(|S|)$ temps si *au moins une* des conditions proposées est satisfaite, e.g. si chaque classe de $P_1$ partage avec une classe de $P_2$ au moins la moitié de leurs éléments. De plus, nous prouvons que l'algorithme peut être très utile dans la pratique pour calculer *presque toutes les distances petite*s. Pour illustrer cela, nous présentons également des conclusions expérimentales observées lors des milliards de calculs de distances exécutés lors de nos recherches heuristiques de colorations. Le nombre de distances inférieures à $\frac{|S|}{10}$ qui n'ont pas pu être calculées en $O(|S|)$ est presque négligeable (quelques centaines sur des milliards, moins de 0.0001%).

L'algorithme proposé pourrait être employé dans de nombreuses autres applications qui nécessitent le traitement des partitions proches. Par exemple, en analyse des données il faut souvent comparer une partition de référence (un "étalon") avec une partition déterminée par un algorithme. Un autre exemple classique est la segmentation d'images, où une partition (une segmentation) obtenue par an algorithme peut être évaluée selon sa distance jusqu'à la segmentation correcte/idéale [Cardoso and Corte-Real, 2005]. En

2. Bien que pour $|S| = 100$ la distance moyenne entre deux partitions aléatoires soit de $69\%|S|$, elle est $66\%|S|$ pour $|S| = 75$, $64\%|S|$ pour $|S| = 50$, etc. Il nous semble que pour $|S| = 1000$ la distance moyenne est d'environ $80\%|S|$.

biologie, cette distance est employée pour estimer la différence (erreur) entre une partition connue d'une population (une structure phylogénique de famille) et une reconstruction bâtie à partir des données génétiques [Konovalov *et al.*, 2005; Berger-Wolf *et al.*, 2007]. Dans le partitionnement de données, on obtient souvent des partitions différentes avec plusieurs algorithmes de clusterisation et il faut trouver un consensus entre ces partitions. Pour cela, on détermine une partition centrale, i.e. une partition qui réduit au minimum la distance moyenne à toutes les autres partitions [da Costa and Rao, 2004; Berman *et al.*, 2007]. S'il n'y pas trop de désaccord entre les algorithmes de clusterisation, les partitions produites seront suffisamment proches pour être manipulées par notre algorithme.

Dans la prochaine section, nous présentons les définitions de base. La Section 6.3 discute des conditions qui permettent de calculer la distance en $O(|S|)$. La Section 6.4 fait une extension de l'algorithme proposé pour le cas où certaines de ces conditions ne sont que partiellement satisfaites. La Section 6.5 discute de l'application de cet algorithme aux calculs de distance entre les colorations de graphe. On termine le chapitre par des conclusions et un exemple de pseudo-code d'algorithme $O(|S|)$.

## 6.2 Définition formelle de la distance

Une $k$-partition $P$ d'un ensemble fini $S = \{1, 2, \ldots, |S|\}$ peut être vue comme une fonction $P : S \to \{1, 2, \ldots, k\}$. Plus souvent, la partition est définie comme un ensemble de classes $\{P^1, P^2, \ldots, P^k\}$ telles que $\bigcup_{1 \leq i \leq k} P^i = S$ et $P^i \cap P^j = \emptyset, \forall i, j \in \{1, 2, \ldots, k\},\ i \neq j$. Les deux définitions sont équivalentes, parce que $P^i = \{x \in S \ : \ P(x) = i\}$ ; $P(x)$ identifie l'indice de la classe de $x$, et ainsi, $x \in P^{P(x)}\ \forall x \in S$. Si la fonction $P$ n'est pas surjective, quelques classes de $P$ sont vides.

Étant données deux $k$-partitions $P_1$ et $P_2$ de $S$, notons $d(P_1, P_2)$ la distance de transfert entre $P_1$ et $P_2$, i.e. le nombre minimum d'éléments qui doivent être transférés entre les classes de $P_1$ afin que la partition résultante soit égale à $P_2$. La similarité $s(P_1, P_2)$ est une mesure complémentaire : le nombre maximum d'éléments de $P_1$ qui *n'ont pas besoin d'être transférés* pour pouvoir transformer $P_1$ en $P_2$. Ainsi, les deux mesures satisfont l'équation suivante :

$$s(P_1, P_2) + d(P_1, P_2) = |S|. \tag{6.1}$$

La distance peut être interprétée comme le nombre minimum d'éléments que l'on doit effacer de $S$, afin que les deux partitions limitées à l'ensemble des éléments restants de $S$ (noté $S'$) soient égales [Gusfield, 2002]. On peut dire que $S'$ représente l'ensemble d'éléments qui sont partagés par les deux partitions, et ainsi, $|S'| = s(P_1, P_2)$. Il est évident que le calcul de la distance est équivalent au calcul de la similarité.

Pour calculer la similarité, on doit trouver une affectation $\sigma : \{1, 2, \ldots, k\} \to \{1, 2, \ldots, k\}$ qui maximise la somme :

$$s(P_1, P_2) = \max_{\sigma} \left( \sum_{1 \leq i \leq k} T_{i,\sigma(i)} \right), \tag{6.2}$$

où $T$ est une matrice $k \times k$ de similarité définie par :

$$T_{ij} = |P_1^i \cap P_2^j| \tag{6.3}$$

Le maximum de cette somme est déterminé en résolvant un problème d'affectation classique. En effet, la plupart des articles [Day, 1981; Gusfield, 2002; Charon *et al.*, 2006; Konovalov *et al.*, 2005] suggèrent juste de déterminer l'affectation maximale $\bar{\sigma}$ en appliquant directement l'algorithme hongrois [Kühn, 1955] sur la matrice de similarité $T$ (qui doit être aussi convertie dans une matrice qui permet de résoudre un problème dual de minimisation).

**Similarité et distance normalisées** Très souvent, il est utile d'employer les valeurs normalisées de la similarité et de la distance : $s_{P_1,P_2} = \frac{s(P_1,P_2)}{|S|}$ et $d_{P_1,P_2} = \frac{d(P_1,P_2)}{|S|}$. Ces valeurs indiquent la proportion d'éléments partagés par deux partitions, $s_{P_1,P_2}$, ou la proportion d'éléments qui doivent être transférés, $d_{P_1,P_2}$. Il est évident – voir (6.1) – , que la formule suivante est toujours satisfaite :

$$d_{P_1,P_2} + s_{P_1,P_2} = 1.$$

Notons que la distance est toujours strictement inférieure à $|S|$, et ainsi, on aurait pu également la normaliser par rapport à d'autres valeurs, par exemple la distance maximum [Charon *et al.*, 2006]. Cependant, la normalisation simple ci-dessus suffit dans cette thèse.

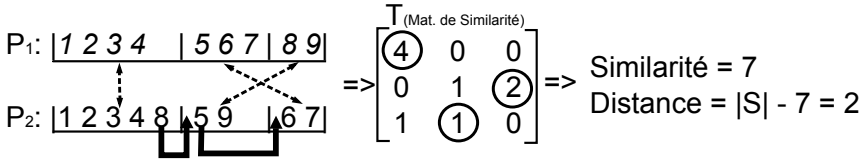


FIGURE 6.1 – Un exemple de calcul de distance. Il faut transférer au moins deux éléments (car $d(P_1, P_2) = 2$) entre les classes d'une partition pour la transformer en l'autre.

La Figure 6.1 illustre le processus de calcul de la distance. Il y a deux 3−partitions $P_1$ et $P_2$ de l'ensemble $S = \{1, 2, \ldots, 9\}$ telles que : $P_1^1 = \{1, 2, 3, 4\}$, $P_1^2 = \{5, 6, 7\}$, et $P_1^3 = \{8, 9\}$ définissent la première partition, et $P_2^1 = \{1, 2, 3, 4, 8\}$, $P_2^2 = \{5, 9\}$ et $P_2^3 = \{6, 7\}$ définissent la deuxième. La matrice de similarité $T$ est calculée avec la formule (6.3) ; la meilleure affectation $\bar{\sigma}$, maximisant la somme dans (6.2), est définie par $\bar{\sigma}(1) = 1$, $\bar{\sigma}(2) = 3$ et $\bar{\sigma}(3) = 2$. Nous obtenons $s(P_1, P_2) = T_{11} + T_{23} + T_{32} = 4 + 2 + 1 = 7$, et ainsi, la distance est $d(P_1, P_2) = |S| - s(P_1, P_2) = 2$ (la distance normalisée est $d_{P_1,P_2} = \frac{2}{9} = 22\%|S|$). En effet, si on change la classe de 2 éléments (e.g. 8 et 5 en $P_2$, cf. Figure 6.1), on transforme une partition en l'autre. .

## 6.3 Calcul de la distance

Dans cette section, nous décrivons la nouvelle approche pour calculer la similarité – et implicitement la distance via (6.1). Nous présentons plusieurs conditions qui pourraient

nous permettre de calculer la similarité entre $P_1$ et $P_2$ en complexité temporelle $O(|S|)$. Notons que si aucune de ces conditions n'est satisfaite, notre algorithme ne perd que $O(|S|)$ pour vérifier cela. L'algorithme comporte deux étapes importantes : (i) la construction de la matrice de similarité $T(P_1, P_2)$, et (ii) la construction du meilleur $\bar{\sigma}$, qui maximise la somme (6.2). Ces deux étapes sont détaillées dans les Sections 6.3.1 et 6.3.2, ci-dessous.

### 6.3.1 Matrice de similarité $T$ en $O(|S|)$

Bien que la matrice $T$ ait $k \times k$ éléments, notre algorithme n'utilise effectivement que $|S|$ valeurs au maximum : les valeurs $T_{ij} = |P_1^i \cap P_2^j|$ avec $i = P_1(x)$ et $j = P_2(x)$, pour tous les $x \in S$. Pour les autres $i$ et $j$ (s'il n'y a pas de $x \in S$ tel que $i = P_1(x)$ et $j = P_2(x)$), la valeur $|P_1^i \cap P_2^j|$ est forcement nulle. La construction de $T$ se fait en trois étapes :

1. L'allocation de mémoire pour T. Bien qu'il s'agisse de $k^2$ éléments, cette opération peut se faire au niveau du *bloc* de mémoire, car il n'y a pas besoin de faire une initialisation – plusieurs instructions en plusieurs systèmes d'exploitations permettent de faire cela en $O(1)$.
2. Parcourir tous les éléments $x \in S$ pour initialiser $T_{P_1(x),P_2(x)} = 0$ en $O(|S|)$ ;
3. Pour chaque élément $x \in S$, incrémenter $T_{P_1(x),P_2(x)} := T_{P_1(x),P_2(x)} + 1$ ; cette opération nécessite $O(|S|)$ opérations.

Dorénavant, les valeurs aux positions $T_{P_1(x),P_2(x)}$ (pour tout $x \in S$) s'appelleront des *éléments actifs*. En fait, la structure matricielle est employée uniquement pour pouvoir indexer ces éléments actifs, pour adresser rapidement les positions $T_{P_1(x),P_2(x)}$ avec $x \in S$. Les autres éléments du $T$ ne sont jamais utilisés, ni pour lire, ni pour écrire. On a tout de même besoin d'une structure de matrice, parce que on ne connaît pas préalablement les positions des éléments actifs.

Concernant l'allocation de la matrice $T$, cela nécessite de trouver un bloc de mémoire qui contient $k^2$ entiers, mais *sans* initialisation de valeur. La plupart des systèmes d'exploitations peuvent faire cela en temps constant – il n'y a aucune contrainte théorique qui impose de parcourir chaque octet du bloc. Nous avons utilisé l'instruction `malloc(`$k^2$`·sizeof(int))`, qui recherche un bloc plus grand que la taille demandée (i.e. au moins $k^2$·sizeof(int) octets) dans la liste des blocs RAM libre [3]. La complexité de cette opération dépend de la fragmentation de la mémoire RAM, et non pas du nombre d'octets demandés. Le lecteur intéressé peut vérifier une implémentation de `malloc` – une des plus populaires est celle de Doug Lea, voir "A memory allocator by Doug Lea". En Windows, la même opération peut se faire via la routine `HeapAlloc()`, avec le drapeau `HEAP_ZERO_MEMORY` désactivé. [4]

3. Il ne faut surtout pas utiliser `calloc`, car cette fonction fait aussi l'initialisation.
4. Nos heuristiques calculent des millions de distances. Même si on fait une allocation-avec-initialisation de complexité $O(k^2)$, cette opération est nécessaire *une seule fois*, parce que la même matrice peut être (re-)utilisée pour tous les calculs de distance (après chaque calcul, on remet à 0 les éléments actifs).

### 6.3.2 Affectation maximale en $O(|S|)$

Cette section présente plusieurs conditions qui nous permettent de calculer la distance de transfert en $O(|S|)$. Nous considérons que l'entrée consiste en un entier $|S|$ (l'ensemble $S$ est toujours $\{1, 2, \dots |S|\}$) et deux vecteurs de taille $|S|$ qui indiquent $P_1(x)$ et $P_2(x)$ pour tous les $x \in S$. Ces deux vecteurs peuvent prendre des valeurs entre 1 et $k$. Nous utilisons la matrice de similarité $T$ construite dans la section précédente et l'objectif est de trouver une affectation maximale $\bar{\sigma}$, i.e. une fonction bijective $\bar{\sigma}$ qui maximise la somme $\sum_{1 \leq i \leq k} T_{i,\sigma(i)}$ dans la formule (6.2).

Pour chaque condition (voir ci-dessous), nous proposons un algorithme Las Vegas – soit il calcule la distance, soit il conclut que la condition n'est pas satisfaite. Autrement dit, le temps "perdu" pour vérifier les conditions est toujours $O(|S|)$ ; il n'accroît pas la complexité totale d'un algorithme plus général si cela s'avère nécessaire.

**Théorème 6.1.** *Si* $\forall i \in \{1, 2, \dots, k\}, \exists j \in \{1, 2, \dots, k\}$ *tel que* $T_{ij} > T_{ij_1}$ *et* $T_{ij} > T_{i_1 j}$, $\forall j_1 \neq j, i_1 \neq i$, *alors la distance de transfert peut être déterminée en* $O(|S|)$.

*Démonstration.* Dans une première étape, il est possible de calculer en $O(|S|)$ temps tous les éléments actifs de $T$ (voir la Section 6.3.1), ainsi que $T_{i\bar{\sigma}(i)}$, l'élément maximum unique de chaque ligne $i$. Pour déterminer ces maximums de ligne, il suffit de parcourir les $O(|S|)$ éléments actifs. Pour chaque élément actif $T_{ij}$, l'algorithme effectue l'instruction suivante : si $T_{ij}$ est plus grand que le maximum courant de la ligne $i$ (au début, ce maximum est zéro pour chaque ligne), le maximum courant est mis à jour, il devient $T_{ij}$.

Comme $T_{i\bar{\sigma}(i)}$ est un maximum strict de la ligne $i$, toute autre fonction $\sigma : S \to S$ conduirait à une plus petite somme $\sum_{1 \leq i \leq k} T_{i,\sigma(i)}$. La vérification du fait que $\bar{\sigma}$ est bijective vient du fait que si $\bar{\sigma}(i) = \bar{\sigma}(i') = j$, alors $T_{ij}$ et $T_{i'j}$ représentent le maximum unique de la colonne $j$ (car chaque maximum strict de ligne doit être aussi un maximum de colonne). Ainsi, $i$ doit être égal à $i'$. □

Les inégalités de ce théorème doivent être strictes, car sinon, il n'est pas toujours possible de déterminer une fonction bijective $\bar{\sigma}$, e.g. si $T = \begin{bmatrix} 2 & 1 \\ 2 & 1 \end{bmatrix}$. Le cas résolu par ce théorème peut être vu comme le dual d'un cas particulier d'algorithme hongrois où le premier pas découvre $k$ zéros mutuellement indépendants (i.e. ne se situant pas sur la même ligne ou colonne). Cependant, nous calculons la distance de transfert sans convertir le problème en un problème de minimisation (l'algorithme hongrois ne résout que des problèmes de minimisation) et sans exécuter les réductions de complexité $O(k \times k)$ requises par l'algorithme hongrois.

Une condition similaire pourrait être exprimée d'une façon plus générique, sans mentionner la matrice $T$.

**Corollaire 6.2.** *Si* $\forall i \in \{1, 2, \dots, k\}, \exists j \in \{1, 2, \dots, k\}$ *tel que* $|P_1^i \cap P_2^j| > \frac{|P_1^i|}{2}$ *et* $|P_1^i \cap P_2^j| > \frac{|P_2^j|}{2}$, *alors la distance de transfert peut être déterminée en* $O(|S|)$.

*Démonstration.* Les conditions de cette hypothèse représentent un cas particulier des conditions du Théorème 6.1. En utilisant (6.3), nous obtenons $\sum_{1 \leq \ell \leq k} T_{i\ell} = \sum_{1 \leq \ell \leq k} |P_1^i \cap$

$P_2^\ell|$. Comme tous les classes $P_2^\ell$ sont disjointes et ont comme union $S$, $\sum_{1\leq\ell\leq k}|P_1^i \cap P_2^\ell| = |P_1^i \cap S| = |P_1^i|$. Par conséquent, pour chaque ligne $i \in \{1, 2, \dots k\}$, les équations suivantes sont valides :

$$\sum_{1\leq\ell\leq k} T_{i\ell} = |P_1^i|. \tag{6.4}$$

Par un raisonnement similaire, nous pouvons conclure que :

$$\sum_{1\leq\ell\leq k} T_{\ell j} = |P_2^j|. \tag{6.5}$$

En utilisant les conditions de l'hypothèse, il s'en suit que $T_{ij} > T_{ij_1}$ et $T_{ij} > T_{i_1 j}$, $\forall j_1 \neq j, i_1 \neq i$ et l'algorithme peut être construit avec le Théorème 6.1. □

L'inconvénient principal des conditions de ce corollaire et du Théorème 6.1 est que, s'il y a une seule ligne $i$ sur laquelle les conditions ne sont *pas* satisfaites, le reste de la construction ne sert plus à notre objectif. Le prochain théorème essaye de contourner ce problème et, par ailleurs, il a l'avantage de ne pas pouvoir être assimilé à une étape duale de l'algorithme hongrois. Nous montrons qu'il est possible de déterminer la meilleure affectation $i \xrightarrow{\bar{\sigma}} j$ sur une ligne particulière $i$ même sans lire toute la matrice. Rappelons que l'algorithme hongrois renvoie uniquement des solutions complètes, il ne peut pas prendre des décisions intermédiaires sur des lignes ou colonnes particulières.

**Théorème 6.3.** *Si pour une ligne $i \in \{1, 2, \dots, k\}$, il y a une colonne $j \in \{1, 2, \dots, k\}$ telle que $T_{ij} \geq T_{ij_1} + T_{i_1 j} \forall j_1 \neq j, i_1 \neq i$, il existe une affectation maximale $\bar{\sigma}$ telle que $\bar{\sigma}(i) = j$. Si le nombre de lignes $i$ qui ne satisfont* pas *cette condition est majoré par $\sqrt[3]{|S|}$, alors la distance de transfert peut être déterminée en $O(|S|)$.*

*Démonstration.* En utilisant un algorithme très similaire à celui du Théorème 6.1, il est possible de construire la matrice $T$, et de déterminer la valeur maximum de chaque ligne et de chaque colonne. En parcourant les $O(|S|)$ éléments actifs, il est possible de marquer les positions de tous les maximums d'une ligne. Ainsi, seulement un élément marqué $T_{ij}$ peut satisfaire $T_{ij} \geq T_{ij_1} + T_{i_1 j} \forall j_1 \neq j, i_1 \neq i$ ; pour vérifier cette condition pour tout maximum $T_{ij}$ d'une ligne, il suffit juste de vérifier que $T_{ij} \geq T_{i,-} + T_{-,j}$, où $T_{i,-}$ et $T_{-,j}$ sont les deuxièmes valeurs maximums de la ligne $i$ et de la colonne $j$, respectivement. Nous considérons que $T_{i,-} = T_{ij}$ si et seulement si la ligne $i$ contient deux (ou plus) éléments de valeur maximum. De plus, la détermination de la deuxième valeur maximum d'une ligne (ou d'une colonne, respectivement) est très similaire à la détermination du premier maximum. Ainsi, la condition de l'hypothèse peut être vérifiée pour toutes les lignes en $O(|S|)$.

Notons $T_{ij}$ un élément marqué par la procédure ci-dessus, tel que $T_{ij} \geq T_{ij_1} + T_{i_1 j} \forall j_1 \neq j, i_1 \neq i$. Nous devons montrer qu'il est possible de construire une affectation maximale qui associe $i$ à $j$. Soit $\sigma$ une affectation maximale. Si $\sigma(i) = j$, alors $\sigma$ est effectivement l'affectation recherchée. Sinon, soit $j_1 = \sigma(i)$ et $i_1 = \sigma^{-1}(j)$ et on obtient :

$$T_{ij} + T_{i_1 j_1} \geq T_{i_1 j} + T_{i j_1} = T_{i_1 \sigma(i_1)} + T_{i\sigma(i)} \tag{6.6}$$

En composant la transposition $(i, i_1)$ avec $\sigma$, on obtient une nouvelle fonction bijective $\bar{\sigma}$ qui diffère de $\sigma$ uniquement sur les positions $i$ et $i_1$ – les valeurs sur ces deux positions sont inversées, i.e. $\bar{\sigma}(i) = j$ et $\bar{\sigma}(i_1) = j_1$. La différence de valeur entre les affectations $\bar{\sigma}$ et $\sigma$ (voir (6.2)) est $T_{ij} + T_{i_1 j_1} - (T_{i_1 j} + T_{i j_1}) \geq 0$. En utilisant (6.6), $\bar{\sigma}$ doit également être une affectation maximale.

Pour récapituler, un algorithme a pu établir une meilleure affectation partielle sur toutes les lignes $i$ qui vérifient la condition de l'hypothèse, indépendamment des lignes où cette condition n'est pas vérifiée. Cette affectation sur ces lignes est déterminée en affectant la ligne $i$ à une ligne $j$ qui satisfait $T_{ij} \geq T_{ij_1} + T_{i_1 j} \forall j_1 \neq j, i_1 \neq i$. Si il y a deux lignes $i_1$ et $i_2$ affectées au même $j$, on affecte que $i_1$ à $j$ et $i_2$ peut être affecté à toute autre valeur de la ligne $i_2$ (parce que toutes ces valeurs doivent être zéro).

Le reste de l'affectation peut être construit en appliquant l'algorithme hongrois sur les lignes et les colonnes restantes. Dans les conditions données en hypothèse, le nombre d'éléments de $\{1, 2, \ldots, k\}$ non affectés par $\bar{\sigma}$ est $k' = \left\lfloor \sqrt[3]{|S|} \right\rfloor$ dans le pire des cas. Pour finir l'affectation, un premier pas marque les $k'$ lignes non affectées et les $k'$ colonnes non affectées. Ensuite, une nouvelle matrice $k' \times k'$ est allouée, et initialisée à zéro en $O(k'^2) < O(|S|)$. Finalement, on parcourt les éléments actifs de $T$ pour copier dans la nouvelle matrice $k' \times k'$ tous les éléments situés à l'intersection d'une ligne et d'une colonne marquées. L'algorithme hongrois détermine la valeur de l'affectation maximale sur cette matrice restreinte en utilisant au maximum $O(k'^3) \leq O(|S|)$ opérations ; ainsi, la complexité totale de notre algorithme est toujours $O(|S|)$. □

Une condition similaire pourrait être exprimée sans mentionner la matrice $T$, d'une façon plus simple.

**Corollaire 6.4.** *Si* $\forall i \in \{1, 2, \ldots, k\}$, $\exists j \in \{1, 2, \ldots, k\}$ *tel que* $|P_1^i \cap P_2^j| \geq \frac{|P_1^i \cup P_2^j|}{2}$, *alors la distance de transfert peut se calculer en* $O(|S|)$.

*Démonstration.* Cette proposition suit des équations (6.4) et (6.5), parce qu'elle devient un cas particulier de Théorème 6.3. Cependant, ce corollaire a l'avantage d'être très facile à implémenter (voir Algorithme 6.1), car $|P_1^i \cup P_2^j| = |P_1^i| + |P_2^j| - |P_1^i \cap P_2^j|$ et $|P_1^i|$ et $|P_2^j|$ peuvent être facilement déterminés.

□

Concernant les conditions les plus générales de cette section, notons que les Théorèmes 6.1 et 6.3 ne peuvent pas résulter l'un de l'autre. Par exemple, si $T = \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix}$, seulement le Théorème 6.1 peut être employé ; si $T = \begin{bmatrix} 2 & 2 \\ 0 & 2 \end{bmatrix}$, on devrait employer le Théorème 6.3. Dans la prochaine section, nous montrons comment le Théorème 6.3 et le Corollaire 6.4 peuvent également être employés dans la pratique pour ne construire qu'une partie de la solution, i.e. une affectation partielle.

## 6.4 Extensions

Dans le cas où les conditions de l'hypothèse du Théorème 6.3 ou du Corollaire 6.4 sont satisfaites seulement par un ensemble restreint de lignes $i \in \{1, 2, \ldots, k\}$, il est toujours possible d'effectuer d'importantes réductions de complexité. Nous prouvons d'abord la proposition suivante :

$$|P_1^i \cap P_2^j| \leq \frac{|P_1^i \cup P_2^j|}{2} \; \forall i, j \in \{1, 2, \ldots, k\} \Longrightarrow s(P_1, P_2) \leq \frac{2}{3}|S|. \tag{6.7}$$

*Démonstration.* Soit $\bar{\sigma}'$ une affectation maximale et $j = \bar{\sigma}'(i)$, où $i$ est une ligne quelconque. On peut écrire $T_{ij} \leq \frac{|P_1^i \cup P_2^j|}{2}$ comme $T_{ij} \leq \frac{|P_1^i| + |P_2^j| - |P_1^i \cap P_2^j|}{2}$, ou $3T_{ij} \leq |P_1^i| + |P_2^j|$. En faisant la somme pour toutes les lignes $i$ ci-dessus, on obtient $3\sum_{1 \leq i \leq k} T_{i\bar{\sigma}'(i)} \leq \sum_{1 \leq i \leq k} |P_1^i| + \sum_{1 \leq j \leq k} |P_2^j| = 2|S|$, et cela prouve (6.7). □

Nous avons utilisé les conditions du Corollaire 6.4 seulement pour une plus grande lisibilité, mais le même résultat peut être déduit des conditions du Théorème 6.3, car si $T_{ij} < T_{ij_1} + T_{i_1 j} \forall j_1 \neq j, i_1 \neq i$, alors $|P_1^i \cap P_2^j| \leq \frac{|P_1^i \cup P_2^j|}{2}$ est également satisfaite – voir (6.4) et (6.5).

Maintenant, nous présentons la réduction effective du calcul de $s(P_1, P_2)$ : nous divisons $S$ en sous-ensembles $A$ et $B$ tels que seul le calcul sur $B$ exige un algorithme d'une complexité plus élevée. Notons $I$ l'ensemble des lignes $i$ tel qu'il existe $j_i \in \{1, 2, \ldots, k\}$ tel que $|P_1^i \cap P_2^{j_i}| > \frac{|P_1^i \cup P_2^{j_i}|}{2}$. Notons $J = \{j \in S : \exists i \in I \text{ tel que } j = j_i\}$ et soit :

$$A = \bigcup_{i \in I} P_1^i \cup P_2^{j_i}, \tag{6.8}$$

et $B = S - A$.

En utilisant $A$ dans le raisonnement du Théorème 6.4 ou 6.3, on trouve qu'il y a une affectation maximale $\bar{\sigma}$ qui satisfait $\bar{\sigma}(i) = j_i, \forall i \in I$. Comme $J$ est l'image de $I$ par la fonction bijective $\bar{\sigma}$, alors $\{1, 2, \ldots, k\} - J$ est l'image de $\{1, 2, \ldots, k\} - I$. Le reste de $\bar{\sigma}$ peut être construit en utilisant uniquement les lignes et les colonnes de ces deux ensembles, qui contiennent des valeurs produites uniquement par des classes de $B$ (i.e. des classes $P_1^i, P_2^j \subseteq B$). [5]

Notons $s(P_1, P_2)|_X$ la similarité entre les partitions $P_1$ et $P_2$ limitées à l'ensemble $X \subset S$. Nous obtenons $s(P_1, P_2) = s(P_1, P_2)|_A + s(P_1, P_2)|_B$. Comme aucune confusion n'est possible, nous pouvons simplement écrire : $|S| \cdot s_{P_1,P_2} = |A| \cdot s_{P_1,P_2}|_A + |B| \cdot s_{P_1,P_2}|_B$ et il est même possible d'omettre l'indice $_{P_1,P_2}$ :

$$s_S = \frac{|A|}{|S|} s_A + \frac{|B|}{|S|} s_B,$$

où $s_X$ est la similarité normalisée entre $P_1$ et $P_2$ limitées à $X$.

5. En effet, les éléments n'appartenant pas à $B$ peuvent être ignorés pour ce calcul car ils sont couverts par une classe de $A$.

Les ensembles $A$ et $B$ peuvent être directement déterminés à partir de l'ensemble $I$ en utilisant (6.8) ; $I$ peut être déterminé en $O(|S|)$ opérations, suivant le raisonnement du Théorème 6.3. De plus, $s_A$ peut être déterminé en $O(|A|) < O(|S|)$, comme nous l'avons expliqué dans la Section 6.3.2 ; $s_B$ peut être déterminé en maximum $O((k-|I|)^3)$ en utilisant l'algorithme hongrois. Pour récapituler, la complexité totale pour calculer la similarité de cette manière est $O(|S|) + O((k-|I|)^3)$ au maximum.

En utilisant (6.7), on obtient $s_B \leq \frac{2}{3}$. Cela signifie que si la similarité totale (sur $S$) est élevée (i.e. par exemple, $s_S > 0.9$), alors $S$ peut être divisée en deux parties :

1. $A$, avec une similarité normalisée $s_A$ très élevée (e.g. $s_A > s_S > 0.9$) qui peut être calculée en $O(|S|)$.
2. $B$, avec une similarité normalisé *nettement* plus petite ($s_B \leq \frac{2}{3}$) et qui ne peut pas être calculée en $O(|S|)$.

Si la similarité totale entre $P_1$ et $P_2$ est élevée (sur $S$), même si nous ne pouvons pas la calculer en $O(|S|)$, nous pouvons toujours identifier en $O(|S|)$ la partie de $S$ (i.e. $A$) telle que la similarité entre les classes de $A$ (entre $P_1$ et $P_2$) soit plus forte. Cela pourrait être particulièrement utile pour les applications qui doivent uniquement trouver les meilleures correspondances/associations entre les classes de deux partitions.

## 6.5 Une étude de cas pour la coloration de graphe

Dans cette section nous présentons des conclusions observées lors de la recherche locale TS–Div (Section 4.2), qui calcule des milliards de distances pour se guider à travers l'espace de recherche. Cet algorithme essaye de rester à une distance minimum de certaines colorations de référence et la complexité du calcul de distance est cruciale – e.g. pour certains graphes, une complexité de $O(k^3)$ pourrait rendre le processus de recherche (au moins) 100 fois plus lent. Les détails de la recherche TS–Div ne sont pas essentiels ici, mais nous présentons une statistique des calculs qu'il a effectué.

Nous considérons un milliard de distances *petites* calculées par TS–Div en résolvant deux instances standard de coloration – i.e. ($dsjc1000.1$, $k = 20$) et ($dsjc1000.5$, $k = 86$) avec 1000 sommets. De nombreuses distances calculées par TS–Div sont *petites*, parce qu'elles sont calculées entre des positions *presque successives* dans une série de colorations voisines. De toute façon, même si TS–Div calcule des distances élevées aussi, nous comptons dans cette statistique uniquement les paires $(P_1, P_2)$ qui satisfont $d(P_1, P_2) < \frac{|S|}{5}$. Pour chaque calcul de distance, TS–Div applique notre approche de calcul, suivant la méthodologie suivante :

1. Si la condition dans le Corollaire 6.4 est satisfaite, l'algorithme calcule la distance correcte en $O(|S|)$ ;
2. Sinon, l'algorithme détecte que la condition n'est pas satisfaite (il retourne ***impossible*** en $O(|S|)$, voir Algorithme 6.1 annexé à la page 111), et, ensuite, l'algorithme hongrois de complexité $O(k^3)$ est exécuté.

La première étape a été suffisante pour plus de 99.99% des cas. Plus précisément, parmi les $10^9$ (petites) distances calculées, nous avons trouvé moins de $10^3$ (i.e. 737 pour $k = 20$

et 880 pour $k = 86$ ) paires $(P_1, P_2)$ telles que $d(P_1, P_2) < \frac{|S|}{5}$ et la condition d'hypothèse du Corollaire 6.4 n'est pas satisfaite. Si nous considérons des distances encore plus petites (plus exactement, seulement les paires $(P_1, P_2)$ telles que $d(P_1, P_2) < \frac{|S|}{10}$) l'algorithme de temps $O(|S|)$ a été suffisant pour tous les cas que nous avons rencontrés.

L'explication de ce succès pratique se trouve dans le fait que la similarité limitée à différentes classes de $S$ présente normalement des valeurs assez homogènes. Ainsi, si la similarité totale est élevée (e.g. si $s_{P_1,P_2} > 0.9$ équivalent à $d(P_1, P_2) < \frac{|S|}{10}$), alors les valeurs de type $s_X$ sont proches de 0.9, pour presque toutes les classes $X$ d'une des partitions. Ainsi, l'ensemble $B$ sur lequel $s_B < \frac{2}{3} = 0.66$ (et les conditions de la Section 6.3.2 ne sont pas satisfaites) est très restreint, généralement vide.

Cependant, théoriquement, il est possible de construire un contre-exemple à cela en prenant deux partitions telles que $P_1^1 = P_2^1$ et $|P_1^1| = 0.9|S|$. Dans ce cas, $s_{P_1,P_2} \geq 0,9$, mais les deux partitions, qui sont très similaires sur $A = P_1^1$, peuvent être totalement différentes sur $B = S - A$. La partie la plus difficile est le calcul de la similarité limitée à $B$ ; cela peut exiger entre $O(k^2)$ et $O(k^3)$ opérations.

## 6.6 Remarques finales

Ce chapitre a présenté un algorithme très rapide pour calculer la distance de transfert entre deux partitions proches $P_1$ et $P_2$ d'un ensemble $S$ (cet ensemble est noté $V$ lorsqu'il s'agit d'un ensemble de sommets dans le reste de la thèse). Nous avons montré qu'il est possible de calculer la distance de transfert en $O(|S|)$ si au moins une condition proposée est satisfaite. En pratique, presque toute distance inférieure à $\frac{|S|}{5}$ peut se calculer en $O(|S|)$. Dans le cadre de la coloration, cet algorithme a permis d'améliorer considérablement la vitesse d'exécution (e.g. le nombre d'itérations par minute) des algorithmes TS–Div et TS–Int. Cela demanderait au moins $O(|S| + k^2)$ opérations et, pour une instance comme ($dsjc1000.9$, $k = 223$) un nombre d'opérations d'environ $|S| + k^2 = 1000 + 223^2 \approx 50.000$ est largement plus coûteux que $|S| = 1000$.

Si aucune condition proposée n'est satisfaite, l'algorithme retourne "impossible" après $O(|S|)$ opérations. Dans cette situation, l'algorithme hongrois est ensuite appliqué, mais la complexité totale de calcul n'est pas alourdie par ces $O(|S|)$ opérations, car $O(|S|)$ compte peu dans une somme de complexités comme $O(|S| + k^2) + O(|S|) = O(|S| + k^2)$. De plus, l'algorithme proposé peut être utile même si les conditions ne sont que partiellement satisfaites sur un sous ensemble de $S$ (voir Section 6.4). Dans une telle situation, il est possible d'identifier le sous-ensemble de $S$ sur lequel les classes sont très similaires, i.e. sur lequel la similarité normalisée est au moins $\frac{2}{3}$. Finalement, certaines idées de l'algorithme proposé peuvent être utiles pour résoudre des problèmes généraux d'affectation, définis avec des matrices creuses, à faible densité.

### 6.6.1 Pseudocode de l'algorithme

L'Algorithme 6.1 présente dans cette annexe un simple pseudocode, correspondant au Corollaire 6.4. En modifiant certains détails, il est possible de transformer cet algorithme

**Algorithme 6.1** : Algorithme de calcul de la distance de transfert, correspondant au Corollaire 6.4

**Entrée :**
- $|S|$ (i.e. $S = \{1, 2, \ldots, |S|\}$)
- $P_1$ et $P_2$ comme $|S|$-vecteurs, i.e. la position $x$ indique $P_1(x)$

**Retour :**
- la distance $\Sigma$, si la condition du Corollaire 6.4 est satisfaite
- *IMPOSSIBLE*, sinon.

1. $\Sigma = 0$
2. $k =$ la valeur maximum dans les vecteurs $P_1$ et $P_2$
3. allocation de matrice $T$ (*sans* initialisation)
4. init. $k-$vecteurs $M$ et $\bar{\sigma}$ à 0 (le maximum sur chaque ligne et l'affectation maximale à construire)
5. init. $k-$vecteurs $|P_1|$ et $|P_2|$ à 0 (cela indique la taille de chaque classe)
6. **For** $x = 1$ **To** $|S|$
   - $i = P_1(x)$ et $j = P_2(x)$
   - $T_{ij} = 0$
7. **For** $x = 1$ **To** $|S|$
   - $i = P_1(x)$ et $j = P_2(x)$
   - incrémenter $T_{ij}$, $|P_1^i|$, $|P_2^j|$
   - **If** $(T_{ij} > M_i)$ **Then** $M_i = T_{ij}$ et $\bar{\sigma}(i) = j$.
8. **For** $i = 1$ **To** $k$
   - **If** $M_i = 0$, **Then Continue** /*avec le prochain $i$ dans la boucle*/
   - **If** $\left(3M_i \leq |P_1^i| + |P_2^{\bar{\sigma}(i)}|\right)$ **Then Return** *IMPOSSIBLE*
   - $\Sigma = \Sigma + T_{i,\bar{\sigma}(i)}$
9. **Return** $\Sigma$

en une version similaire qui correspond aux autres théorèmes introduits dans ce chapitre. Le nombre de classes n'a pas besoin d'être le même pour $P_1$ et $P_2$, parce que $k$ n'est pas donné en entrée ; $k$ est calculé comme le nombre maximum des classes dans une des deux partitions (voir Pas 2).

# Conclusion Générale

## Contributions

Nous avons proposé plusieurs approches algorithmiques pour introduire une forme d'apprentissage dans les stratégies de recherche, dans le but général de rendre les heuristiques classiques "mieux informées". Pour cela, nous avons utilisé des techniques de fouille de données (e.g. *Multidimensional Scaling*) ; de plus, nous avons toujours tenté de fouiller et exploiter les informations les plus pertinentes pour tout composant d'un algorithme. Quatre algorithmes ont été testés dans le cadre expérimental très concurrentiel de la coloration de graphe. En effet, la coloration est l'un des plus étudiés et des plus connus des problèmes $NP$-complets, et ainsi, une dizaine de nouveaux algorithmes heuristiques a été publiée uniquement pendant cette thèse de trois ans.

Selon l'ordre chronologique de déroulement de thèse, notre premier apport concerne un algorithme Tabou de base, qui a été amélioré avec de nouvelles fonctions d'évaluation "bien informées". En couplant quelques techniques algorithmiques additionnelles – i.e. une liste Tabou réactive – cette recherche Tabou non sophistiquée peut atteindre des résultats très performants par rapport à plusieurs algorithmes plus complexes.

Par la suite, l'analyse de la trajectoire de cette recherche locale nous a conduits (Chapitre 3) à l'hypothèse de clusterisation : les meilleures configurations ne sont pas uniformément dispersées dans l'espace de recherche, mais elles sont groupées en clusters qui peuvent être confinés dans des sphères de rayon $R = 10\%|V|$. Dans le reste de la thèse, nous avons considéré que deux colorations distantes de moins de $R = 10\%|V|$ sont "trop proches" – il est très facile de passer de l'une à l'autre. Le rayon $R$ est défini en utilisant la distance de transfert entre partitions ; cette distance est équivalente au nombre minimum de transitions de voisinage entre colorations, pour notre problème et pour notre voisinage.

Au quatrième chapitre, nous avons présenté deux recherches locales guidées, fondées sur cette distance et sur la notion de sphère. Le premier algorithme (TS–Div) "est guidé" vers la diversification. Il mémorise sa propre trajectoire à travers l'espace de recherche en enregistrant les sphères visitées. De cette manière, il détecte l'instant où le processus de recherche entre dans une sphère déjà visitée. Quand cela se produit, TS–Div agit afin d'induire plus de diversification, ce qui permet de quitter plus rapidement cette sphère, et d'aller continuellement vers de nouvelles régions. Cela peut garantir que TS–Div n'effectue pas trop d'explorations redondantes à long terme. Comme la liste Tabou interdit des explorations redondantes à court terme, TS–Div peut couvrir uniformément l'espace de recherche.

L'algorithme TS–Int est guidé pour l'intensification, afin d'explorer minutieusement un *périmètre limité* donné, une région prometteuse spécifique. À partir d'une configuration de départ, il lance plusieurs processus de recherche locale qui explorent *uniquement l'intérieur de la sphère* de la configuration de départ. Un tel processus est arrêté dès qu'il sort de cette sphère, et "le point de sortie de sphère" est inséré dans une file contenant des centres des sphères à explorer plus tard. Quand les processus lancés à partir d'un centre de sphère ne

trouvent plus de configurations distantes de grande qualité (l'investigation de la sphère est complète), le prochain centre est considéré et TS–Int continue avec une autre investigation de sphère (autour de cette nouvelle configuration). Le résultat est un parcours en largeur de sphères d'un périmètre limité ; cet algorithme semble capable d'atteindre toute solution sur une distance de $30\%|V|$ du point de départ, avec un taux de succès empirique de 100%.

Une approche évolutionniste est présentée au Chapitre 5. Une nouveauté est l'introduction d'une stratégie d'écart qui empêche la population de garder en même temps deux individus de la même sphère. En utilisant des mécanismes de dispersion réactive à base de distances, un écart approprié entre les individus est toujours assuré. De cette façon, la convergence prématurée est toujours évitée ; l'algorithme Evo–Div peut effectivement créer continuellement de la diversité utile, et cela sans sacrifice de qualité. Un croisement "bien informé" est également présenté ; il emploie plusieurs informations sur les classes de couleur des parents afin de choisir les meilleurs gènes à passer à la génération suivante.

Au dernier chapitre, la distance de transfert est étudiée en détail, fournissant plus d'informations sur l'interprétation des valeurs de distance. Nous introduisons également un algorithme exact de type *Las Vegas* pour calculer efficacement cette distance. Il permet de réduire la complexité de $O(|V| + k^3)$ à $O(|V| + k)$ si au moins une condition présentée est vérifiée. Cet algorithme a apporté une importante accélération à TS–Div, et, de plus, il pourrait être employé dans d'autres applications où il faut calculer des distances entre des partitions proches.

Au niveau des résultats pratiques, bien que le nombre d'algorithmes de coloration ait accru continuellement ces dernières années, nous avons atteint toutes les meilleures bornes supérieures connues (sauf pour le graphe $r1000.5$) et nous avons même trouvé une nouvelle borne supérieure pour un grand graphe ($C4000.5$, $k = 271$). [6] Il pourrait être intéressant de noter quelques bornes supérieures particulièrement difficiles :

- ($latin\_square$, $k = 98$), nous avons trouvé cette solution (avec Evo–Div) pour la première fois en presque *15 ans* – jusqu'à maintenant, une solution pour cette instance a été rapportée uniquement dans [Morgenstern, 1996] ;
- ($dsjc1000.9$, $k = 223$), la thèse introduit même algorithmes qui peuvent atteindre cette borne (soit Evo-Div, soit TS–Int avec TS–Div) – un seul autre algorithme a trouvé une solution très récemment [Lü and Hao, 2010] ;
- ($flat300.28$, $k = 28$), TS–Int, en combinaison avec TS–Div, peut atteindre assez facilement cette borne – à ce jour, ce résultat avait été rapporté uniquement dans [Blöchliger and Zufferey, 2008; Hertz *et al.*, 2008] ;
- ($r1000.5$, $k = 237$), bien que ce graphe soit le seul pour lequel nous n'ayons pas atteint la meilleure borne connue, notons qu'il existe très peu d'algorithmes [Prestwich, 2002; Malaguti *et al.*, 2008] qui ont trouvé le nombre chromatique (234) et ils ont utilisé des techniques exactes. Les autres algorithmes purement heuristiques ne sont pas descendus sous la barre de $k = 245$.

6. Toutes les colorations légales mentionnées dans cette thèse sont disponibles en ligne : `info.univ-angers.fr/pub/porumbel/graphs/bestcol/`

## Perspectives

L'idée d'utiliser des distances pour construire une *recherche guidée* semble, dans une certaine mesure, négligée dans l'optimisation combinatoire. Des techniques à base des distances ont été appliquées uniquement pour des taches très spécifiques (e.g. croisements de type *distances-preserving*, indices de difficulté ou de corrélation *fitness-distance*). Cependant, nous pensons que les distances peuvent être employées d'une manière systématique, à la fois pour comprendre l'espace de recherche, et aussi pour équiper un processus de recherche avec une "boussole" de navigation/orientation dans l'espace de recherche.

Bien que ces idées n'aient été mises en œuvre et validées que sur le problème de coloration de graphe, certaines techniques sont très générales et pourraient également être testées sur d'autres problèmes. Par exemple, comment peut-on savoir si une recherche locale d'un problème quelconque couvre uniformément l'espace de recherche, ou si elle exécute plutôt des explorations redondantes ? L'enregistrement de granularité grossière effectué par TS–Div offre une solution potentielle : en enregistrant un ensemble restreint de sphères, on peut effectivement inspecter l'évolution de tout processus de recherche et clarifier ce type de questions. La seule condition pour réaliser cela est de pouvoir calculer une distance de voisinage, i.e. une distance qui indique le nombre minimum de transitions de voisinage entre deux configurations – voir plusieurs exemples dans la Section 4.5.

Concernant l'approche évolutionniste, il serait très intéressant de formaliser une stratégie générique d'écart : nous pensons que les idées de contrôle d'écart peuvent être directement utilisées dans d'autres algorithmes mémétiques. Évidement, pour arriver aux meilleurs résultats pratiques, il est toujours essentiel d'étudier les détails et les particularités spécifiques du problème, afin d'adapter l'approche. Cependant, comme cela a été discuté en Section 5.4.4, des idées similaires ont été déjà testées avec succès dans d'autres communautés évolutionnistes (e.g. optimisation continue multimodale), et ainsi, nous pensons que d'autres progrès importants sont à prévoir.

De plus, un travail plus en profondeur pourrait coupler les idées concernant l'enregistrement du chemin de la recherche locale avec le contrôle d'écart dans l'algorithme évolutionniste. Tandis que le contrôle d'écart assure qu'il n'y a pas simultanément deux individus trop proches dans la population, il n'assure pas de propriété similaire sur des individus appartenant à des générations différentes – ce type de problème pourrait être évité en enregistrant les sphères des individus à toutes les générations.

# Liste des figures

1.1 Un exemple d'un enregistrement à gros grain du chemin d'exploration . . . . . 12

2.1 Deux colorations avec le même nombre de conflits mais avec potentiel différent 29
2.2 Évolution typique du nombre de conflits avec plusieurs fonctions d'évaluation . 34
2.3 Histogrammes du nombre de conflits obtenus avec 1000 descentes pures . . . . 36

3.1 Représentation 3D de 350 optima de l'espace multidimensionnel d'un problème 48
3.2 Colorations de grande qualité visitées par une recherche locale . . . . . 49
3.3 Histogrammes des distances entre chaque couple des 40,000 configurations de grande qualité visitées par une recherche locale . . . . . 51

4.1 Exemple schématique d'un processus TS–Div explorant l'espace de recherche . 56
4.2 Représentation MDS d'une évolution réelle de TS–Int . . . . . 64

6.1 Un exemple de calcul de distance . . . . . 103

# Liste des tableaux

1.1 Bornes supérieures DIMACS faciles . . . . . 18

2.1 Moyenne du nombre de conflits après les 1.000.000 premières itérations avec $f$, $\widetilde{f}_1$ et $\widetilde{f}_2$ . . . . . 35
2.2 Résultats détaillés de RCTS avec un temps limite de 10 heures . . . . . 38
2.3 Résultats du RCTS et des meilleurs algorithmes les plus performants . . . . . 40

4.1 Temps et mémoire consommée par la recherche guidé TS–Div . . . . . 61
4.2 Les résultats d'algorithme TS–Div guidé vers la diversification . . . . . 66
4.3 Les résultats d'algorithme TS–Int guidé vers l'intensification . . . . . 67
4.4 Comparaison entre TS–Div/TS–Int et les meilleurs algorithmes . . . . . 73

5.1 Résumé des paramètres génétiques. . . . . 91
5.2 Résultats détaillés d'Evo–Div avec un temps (CPU) limite de 300 minutes . . . 92
5.3 Les résultats d'Evo–Div avec deux limites de temps . . . . . 93
5.4 Les meilleures colorations trouvées par Evo–Div dans moins que 5 heures et les résultats des meilleurs algorithmes . . . . . 95
5.5 Comparaison d'Evo–Div standard avec cinq versions différentes . . . . . 96

# Liste des algorithmes

2.1 La recherche Tabou pour $k$-Coloration . . . . . 27
4.1 La (méta) heuristique TS–Div . . . . . 58
4.2 Pseudo-code TS–Int . . . . . 62
5.1 Schéma de Base de l'Algorithme Évolutionniste Evo–Div . . . . . 78
5.2 Croisement "bien informé" entre partitions . . . . . 81
5.3 La fonction de remplacement (élimination) d'Evo–Div . . . . . 87
6.1 Algorithme de calcul de la distance de transfert . . . . . 111

# Références bibliographiques


[Avanthay *et al.*, 2003] cité page 14, 19
C. Avanthay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151(2) :379–388, 2003.

[Barnier and Brisset, 2004] cité page 13
N. Barnier and P. Brisset. Graph coloring for air traffic flow management. *Annals of Operations Research*, 130(1) :163–178, 2004.

[Battiti and Tecchiolli, 1994] cité page 9
R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6 :126–126, 1994.

[Battiti *et al.*, 2008] cité page 6, 8, 9, 9, 32, 43, 57
R. Battiti, R Brunato, and F. Mascia. *Reactive Search and Intelligent Optimization*. Springer, 2008.

[Beasley *et al.*, 1993] cité page 89, 89, 89
D. Beasley, D.R. Bull, and R.R. Martin. A sequential niche technique for multimodal function optimization. *Evolutionary Computation*, 1(2) :101–125, 1993.

[Bellare *et al.*, 1998] cité page 12
M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCPs, and nonapproximability-towards tight results. *SIAM Journal on Computing*, 27(3) :804, 1998.

[Berger-Wolf *et al.*, 2007] cité page 101, 101
T.Y. Berger-Wolf, S.I. Sheikh, B. DasGupta, M.V. Ashley, I.C. Caballero, W. Chaovalitwongse, and S.L. Putrevu. Reconstructing sibling relationships in wild populations. *Bioinformatics*, 23(13) :i49–56, 2007.

[Berman *et al.*, 2007] cité page 101, 101
P. Berman, B. DasGupta, M.Y. Kao, and J. Wang. On constructing an optimal consensus clustering from multiple clusterings. *Information Processing Letters*, 104(4) :137–145, 2007.

[Blöchliger, 2005] cité page 13
I. Blöchliger. *Suboptimal Colorings and Solution of Large Chromatic Scheduling Problems*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2005.

[Blöchliger and Zufferey, 2008] cité page 13, 14, 14, 15, 15, 20, 24, 25, 32, 32, 37, 44, 69, 70, 70, 92, 93, 98, 114

I. Blöchliger and N. Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers and Operations Research*, 35(3) :960–975, 2008.

[Boese *et al.*, 1994] cité page 10
KD Boese, AB Kahng, and S. Muddu. A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 16(2) :101–113, 1994.

[Bouziri *et al.*, 2008] cité page 20
H. Bouziri, E.G. Talbi, and K. Mellouli. A cooperative search method for the k-coloring problem. *Journal of Mathematical Modelling and Algorithms*, 7(2) :125–142, 2008.

[Boyan *et al.*, 2000] cité page 8, 9
J. Boyan, W. Buntine, and A. Jagota. Statistical machine learning for large-scale optimization. *Neural Computing Surveys*, 3(1) :1–58, 2000.

[Brelaz, 1979] cité page 13, 14
D. Brelaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4) :251–256, 1979.

[Brown, 1972] cité page 13
J.R. Brown. Chromatic scheduling and the chromatic number problem. *Management Science*, 19(4) :456–463, 1972.

[Burke *et al.*, 1994] cité page 13
E. K. Burke, D. G. Elliman, and R. F. Weare. A university timetabling system based on graph colouring and constraint manipulation. *Journal of Research on Computing in Education*, 27(1) :1–18, 1994.

[Cardoso and Corte-Real, 2005] cité page 100, 101, 101
J.S. Cardoso and L. Corte-Real. Toward a generic evaluation of image segmentation. *IEEE Transactions on Image Processing*, 14(11) :1773–1782, 2005.

[Carpaneto and Toth, 1983] cité page 100
G. Carpaneto and P. Toth. Algorithm for the solution of the assignment problem for sparse matrices. *Computing*, 31(1) :83–94, 1983.

[Cedeño and Vemuri, 1999] cité page 89, 89, 89
W. Cedeño and V.R. Vemuri. Analysis of speciation and niching in the multi-niche crowding ga. *Theoretical Computer Science*, 229(1) :177, 1999.

[Chaitin, 1982] cité page 13
G. J. Chaitin. Register allocation & spilling via graph coloring. In *Proceedings of the 1982 SIGPLAN symposium on Compiler construction*, pages 98–105, New York, NY, USA, 1982. ACM.

[Chams *et al.*, 1987] cité page 14
M. Chams, A. Hertz, and D. de Werra. Some experiments with simulated annealing for coloring graphs. *European Journal of Operational Research*, 32(2) :260–266, 1987.

[Charon *et al.*, 2006] cité page 100, 101, 103, 103
I. Charon, L. Denoeud, A. Guénoche, and O. Hudry. Maximum transfer distance between partitions. *Journal of Classification*, 23(1) :103–121, 2006.

[Chiarandini and Stützle, 2002] cité page 14, 19
M. Chiarandini and T Stützle. An application of iterated local search to graph coloring. In D. S. Johnson et al., editors, *Computational Symposium on Graph Coloring and its Generalizations*, pages 112–125, 2002.

[Chiarandini *et al.*, 2007] cité page 13
M. Chiarandini, I. Dumitrescu, and T. Stützle. Stochastic local search algorithms for the graph colouring problem. In T.F. Gonzalez, editor, *Handbook of approximation algorithms and metaheuristics*, pages 63–1–63–17. Chapman & Hall/CRC, Boca Raton, FL, USA., 2007.

[Chiarandini, 2005] cité page 13
M. Chiarandini. *Stochastic Local Search Methods for Highly Constrained Combinatorial Optimisation Problems*. PhD thesis, Darmstadt University of Technology, Germany,, 2005.

[Chow and Hennessy, 1990] cité page 13
F.C. Chow and J.L. Hennessy. The priority-based register allocation. *ACM Transactions on Programming Languages and Systems*, 12(4) :501–536, 1990.

[Christofides, 1971] cité page 13
N. Christofides. An algorithm for the chromatic number of a graph. *The Computer Journal*, 14(1) :38–39, 1971.

[Coello *et al.*, 2004] cité page 88
C.A.C. Coello, G.T. Pulido, and M.S. Lechuga. Handling multiple objectives with particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3), 2004.

[Costa and Hertz, 1997] cité page 14
D. Costa and A. Hertz. Ants can colour graphs. *Journal of the Operational Research Society*, 48(3) :295–305, 1997.

[Costa *et al.*, 1995] cité page 77
D. Costa, A. Hertz, and C. Dubuis. Embedding a sequential procedure within an evolutionary algorithm for coloring problems in graphs. *Journal of Heuristics*, 1(1) :105–128, 1995.

[Culberson and Gent, 2001] cité page 42, 43
J.C. Culberson and I. Gent. Frozen development in graph coloring. *Theoretical Computer Science*, 265 :227–264, 2001.

[Culberson and Luo, 1996] cité page 14, 17, 68
J.C. Culberson and F. Luo. Exploring the k-colorable landscape with iterated greedy. In *Cliques, Coloring, and Satisfiability Second DIMACS Implementation Challenge* [Johnson and Trick, 1996], pages 345–284.

[da Costa and Rao, 2004] cité page 100, 101
J.F.P. da Costa and P.R. Rao. Central partition for a partition-distance and strong pattern graphs. *REVSTAT–Statistical Journal*, 2(2) :127–143, 2004.

[Day, 1981] cité page 100, 103
W.H.E. Day. The complexity of computing metric distances between partitions. *Mathematical Social Sciences*, 1 :269–287, 1981.

[De Jong, 1975] cité page 89, 89
K.A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan Ann Arbor, MI, USA, 1975.

[De Werra and Gay, 1994] cité page 13
D. De Werra and Y. Gay. Chromatic scheduling and frequency assignment. *Discrete Applied Mathematics*, 49(1-3) :165–174, 1994.

[de Werra, 1985] cité page 13
D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2) :151–162, 1985.

[Deb and Goldberg, 1989] cité page 89, 89
K. Deb and D. E. Goldberg. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 42–50, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[Deb *et al.*, 2002] cité page 88, 88
K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2) :182–197, 2002.

[Denœud and Guénoche, 2006] cité page 100, 101
L. Denœud and A. Guénoche. Comparison of distance indices between partitions. In V. Batagelj et al., editors, *Data Science and Classification*, pages 21–28. Springer, Berlin, Germany, 2006.

[Denœud, 2008] cité page 101, 101, 101
L. Denœud. Transfer distance between partitions. *Data Science and Classification*, 2(3) :279–294, 2008.

[Devarenne *et al.*, 2006] cité page 14, 19, 25, 31, 32, 32
I. Devarenne, Mabed H., and A. Caminada. Intelligent neighborhood exploration in local search heuristics. In *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence*, pages 144–150. IEEE Computer Society, 2006.

[Devarenne, 2007] cité page 13
I. Devarenne. *Études en recherche locale adaptative pour l'optimisation combinatoire.* PhD thesis, Université de Technologie de Belfort Montbéliard, France, 2007.

[Dorne and Hao, 1995] cité page 13
R. Dorne and J.K. Hao. An evolutionary approach for frequency assignment in cellular radio networks. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 539–544, 1995.

[Dorne and Hao, 1998a] cité page 14, 14, 15, 20, 24, 25, 37, 76, 77, 82
R. Dorne and J.K. Hao. A new genetic local search algorithm for graph coloring. In *PPSN 98*, volume 1498 of *LNCS*, pages 745–754. Springer, 1998.

[Dorne and Hao, 1998b] cité page 14, 14, 25, 25, 33
R. Dorne and J.K. Hao. Tabu search for graph coloring, t-colorings and set t-colorings. In S. Voss et al., editors, *Meta-Heuristics Advances and Trends in Local Search Paradigms for Optimization*, pages 77–92. Kluwer, 1998.

[Dorne, 1998] cité page 13
R. Dorne. *Méthodes heuristiques pour la coloration, la T-coloration et l'affectation de fréquences*. PhD thesis, Université de de Montpellier, France, 1998.

[Du and Pardalos, 2007] cité page 42, 60
D. Du and P.M. Pardalos. *Handbook of Combinatorial Optimization*. Springer, 2007.

[Falkenauer, 1998] cité page 76
E. Falkenauer. *Genetic algorithms and grouping problems*. John Wiley & Sons, Inc. New York, NY, USA, 1998.

[Fleurent and Ferland, 1996a] cité page 14, 14, 14, 24, 25, 25, 26, 77
C. Fleurent and J.A. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63(3) :437–461, 1996.

[Fleurent and Ferland, 1996b] cité page 20, 82
C. Fleurent and J.A. Ferland. Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. In *Cliques, Coloring, and Satisfiability Second DIMACS Implementation Challenge* [Johnson and Trick, 1996], pages 619–652.

[Fotakis *et al.*, 2001] cité page 20
D.A. Fotakis, S.D. Likothanassis, and S.K. Stefanakos. An evolutionary annealing approach to graph coloring. *Proceedings of EvoWorkshops2001—Applications of Evolutionary Computing*, 2037 :120–129, 2001.

[Frawley *et al.*, 1992] cité page 1
W.J. Frawley, G. Piatetsky-Shapiro, and C.J. Matheus. Knowledge discovery in databases : An overview. *AI Magazine*, 13(3) :57–70, 1992.

[Freisleben and Merz, 1996] cité page 88
B. Freisleben and P. Merz. A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 616–621, 1996.

[Funabiki and Higashino, 2000] cité page 19
N. Funabiki and T. Higashino. A minimal-state processing search algorithm for graph coloring problems. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 83(7) :1420–1430, 2000.

[Galinier and Hao, 1999] cité page 14, 14, 14, 15, 20, 24, 25, 25, 33, 76, 77, 82, 88, 96
P. Galinier and J.K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4) :379–397, 1999.

[Galinier and Hertz, 2006] cité page 13, 14, 25
P. Galinier and A. Hertz. A survey of local search methods for graph coloring. *Computers and operations research*, 33(9) :2547–2562, 2006.

[Galinier *et al.*, 2008] cité page 14, 15, 20, 24, 77, 82, 92, 98
P. Galinier, A. Hertz, and N. Zufferey. An adaptive memory algorithm for the k-coloring problem. *Discrete Applied Mathematics*, 156(2) :267–279, 2008.

[Galinier, 1999] cité page 13, 26
P. Galinier. *Etude des métahueristiques pour la satisfaction de contraintes et la coloration.* PhD thesis, Université de Montpellier, France, 1999.

[Gamache *et al.*, 2007] cité page 13
M. Gamache, A. Hertz, and J.O. Ouellet. A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding. *Computers and Operations Research*, 34(8) :2384–2395, 2007.

[Gamst and Rave, 1982] cité page 13
A. Gamst and W. Rave. On frequency assignment in mobile automatic telephone systems. In *Proceedings of the IEEE Global Communications Conference*, pages 309–315, 1982.

[Garey *et al.*, 1979] cité page 6, 6, 12
M.R. Garey, D.S. Johnson, et al. *Computers and Intractability : A Guide to the Theory of NP-completeness.* Freeman, San Francisco, U.S.A., 1979.

[Gebremedhin *et al.*, 2005] cité page 13
A.H. Gebremedhin, F. Manne, and A. Pothen. What color is your jacobian ? graph coloring for computing derivatives. *SIAM Review*, 47(4) :629, 2005.

[Gerber *et al.*, 1998] cité page 42
M.U. Gerber, P. Hansen, and A. Hertz. Local optima topology for the 3-sat problem. *Cahiers du GERAD*, G–98–68, 1998.

[Glass and Pruegel-Bennett, 2005] cité page 59, 59
C.A. Glass and A. Pruegel-Bennett. A polynomially searchable exponential neighbourhood for graph colouring. *Journal of the Operational Research Society*, 56(3) :324–330, 2005.

[Glass, 2002] cité page 13
CA Glass. Bag rationalisation for a food manufacturer. *Journal of the Operational Research Society*, pages 544–551, 2002.

[Glover and Laguna, 1997] cité page 23
F. Glover and M. Laguna. *Tabu Search.* Springer, 1997.

[Glover *et al.*, 1996] cité page 20, 30
F. Glover, M. Parker, and J. Ryan. Coloring by tabu branch and bound. In *Cliques, Coloring, and Satisfiability Second DIMACS Implementation Challenge* [Johnson and Trick, 1996], pages 285–307.

[Glover, 1986] cité page 7, 23
F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations research*, 13(5) :533–549, 1986.

[Goldberg and Richardson, 1987] cité page 89, 89, 89
D.E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal

function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. L. Erlbaum Associates Inc. Hillsdale, NJ, USA, 1987.

[Gusfield, 2002] cité page 59, 59, 59, 100, 102, 103
D. Gusfield. Partition-distance a problem and class of perfect graphs arising in clustering. *Information Processing Letters*, 82(3) :159–164, 2002.

[Hale, 1980] cité page 13
W.K. Hale. Frequency assignment theory and applications. *Proceedings of the IEEE*, 68(12) :1497–1514, 1980.

[Hamiez and Hao, 2001] cité page 20, 82, 87
J. P. Hamiez and J. K. Hao. Scatter search for graph coloring. In *Artificial Evolution*, volume 2310 of *LNCS*, pages 168–179. Springer, 2001.

[Hamiez and Hao, 2004] cité page 20, 42, 43
J.P. Hamiez and J.K. Hao. An analysis of solution properties of the graph coloring problem. In *Metaheuristics computer decision-making*, pages 325–345. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[Hansen, 1986] cité page 23
P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy*, 1986.

[Hao *et al.*, 1999] cité page 6
J.K. Hao, P. Galinier, and M. Habib. Metaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Revue d'Intelligence Artificielle*, 13(2) :283–324, 1999.

[Hertz and Werra, 1987] cité page 14, 14, 23, 25
A. Hertz and D. Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4) :345–351, 1987.

[Hertz *et al.*, 1994] cité page 27, 42, 43, 43
A. Hertz, B. Jaumard, and M.P. de Aragão. Local optima topology for the k-coloring problem. *Discrete Applied Mathematics*, 49(1-3) :257–280, 1994.

[Hertz *et al.*, 2008] cité page 14, 15, 15, 20, 24, 37, 44, 69, 70, 70, 92, 93, 98, 114
A. Hertz, A. Plumettaz, and N. Zufferey. Variable space search for graph coloring. *Discrete Applied Mathematics*, 156(13) :2551–2560, 2008.

[Hoos and Stützle, 2004] cité page 6, 7, 8, 9, 17
H. Hoos and T. Stützle. *Stochastic Local Search Foundations & Applications*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2004.

[Horn *et al.*, 1994] cité page 43
J. Horn, D.E. Goldberg, and K. Deb. Long path problems. In *PPSN 94*, volume 866 of *LNCS*, pages 149–158. Springer, 1994.

[Johnson and Trick, 1996] cité page 13, 15, 17, 123, 125, 126
D.S. Johnson and M. Trick. *Cliques, Coloring, and Satisfiability Second DIMACS Implementation Challenge*, volume 26 of *DIMACS series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1996.

[Johnson *et al.*, 1991] cité page 14, 14, 17, 17, 18, 31, 72
D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing an experimental evaluation ; part ii, graph coloring and number partitioning. *Operations Research*, 39(3) :378–406, 1991.

[Johnson *et al.*, 2002] cité page 15
D. Johnson, A. Mehrotra, and M. Trick. Computational Symposium on Graph Coloring and Its Generalizations, COLOR02, Cornell University, Ithaca, NY, 2002. (`mat.gsia.cmu.edu/COLORING02/`).

[Johnson *et al.*, 2008] cité page 13, 15
D.S. Johnson, A. Mehrotra, and M.A. Trick. Special issue on computational methods for graph coloring and its generalizations. *Discrete Applied Mathematics*, 156(2) :145–146, 2008.

[Jones and Forrest, 1995] cité page 42, 43
T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192, 1995.

[Kallel *et al.*, 2001] cité page 42
L. Kallel, B. Naudts, and C. R. Reeves. Properties of fitness functions and search landscapes. In *Theoretical aspects of evolutionary computing*, pages 175–206. Springer-Verlag, London, UK, 2001.

[Karp, 1972] cité page 12
R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[Kauffman and Levin, 1987] cité page 43
S. Kauffman and S. Levin. Towards a general theory of adaptive walks on rugged landscapes. *Journal of theoretical Biology*, 128(1) :11, 1987.

[Kendall, 1938] cité page 71
MG Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2) :81–93, 1938.

[Konovalov *et al.*, 2005] cité page 100, 101, 103
D.A. Konovalov, B. Litow, and N. Bajema. Partition-distance via the assignment problem. *Bioinformatics*, 21(10) :2463–2468, 2005.

[Kruskal, 1964] cité page 46, 47
J.B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1) :1–27, 1964.

[Kühn, 1955] cité page 100, 103
H.W. Kühn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2 :83–97, 1955.

[Kuntz *et al.*, 2004] cité page 42
P. Kuntz, B. Pinaud, and R. Lehn. Elements for the description of fitness landscapes associated with local operators for layered drawings of directed graphs. In *Metaheuristics*

*Computer Decision-Making*, pages 405–420. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[Laurent and Hao, 2009] cité page 13
B. Laurent and J.K. Hao. List-graph colouring for multiple depot vehicle scheduling. *International Journal of Mathematics in Operational Research*, 1(1) :228–245, 2009.

[Leighton, 1979] cité page 13, 18, 69
F.T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6) :489–503, 1979.

[Lerman *et al.*, 1981] cité page 7
I.C. Lerman, T. Chantrel, and I. Cohen. *Classification et analyse ordinale des données.* Dunod Paris, 1981.

[Lewandowski and Condon, 1996] cité page 13, 13, 18, 18
G. Lewandowski and A. Condon. Experiments with parallel graph coloring heuristics and applications of graph coloring. In *Cliques, Coloring, and Satisfiability Second DIMACS Implementation Challenge* [Johnson and Trick, 1996], pages 309–334.

[Lim and Wang, 2005] cité page 13
A. Lim and F. Wang. Robust graph coloring for uncertain supply chain management. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05)-Track 3-Volume 03*. IEEE Computer Society, 2005.

[Lü and Hao, 2009] cité page 14, 20
Z. Lü and J.K. Hao. A critical element-guided perturbation strategy for iterated local search. In *EvoCOP*, volume 5482 of *LNCS*, pages 1–12, 2009.

[Lü and Hao, 2010] cité page 13, 14, 15, 20, 20, 39, 66, 77, 82, 94, 98, 114
Z. Lü and J.K. Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(2) :241–250, 2010.

[Lucet *et al.*, 2006] cité page 13
C. Lucet, F. Mendes, and A. Moukrim. An exact method for graph coloring. *Computers and Operations Research*, 33(8) :2189–2207, 2006.

[Lund and Yannakakis, 1994] cité page 12
C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5) :960–981, 1994.

[Mahfoud, 1995a] cité page 89, 89, 89
S.W. Mahfoud. A comparison of parallel and sequential niching methods. In L. J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 136–143, 1995.

[Mahfoud, 1995b] cité page 89, 89
S.W. Mahfoud. *Niching methods for genetic algorithms.* PhD thesis, University of Illinois at Urbana-Champaign, 1995.

[Malaguti and Toth, 2008] cité page 14, 82, 82
E. Malaguti and P. Toth. An evolutionary approach for bandwidth multicoloring problems. *European Journal of Operational Research*, 189(3) :638–651, 2008.

[Malaguti and Toth, in press] cité page 6, 13, 13, 13
E. Malaguti and P. Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, in press. `http://dx.doi.org/10.1111/j.1475-3995.2009.00696.x`.

[Malaguti *et al.*, 2008] cité page 13, 14, 15, 20, 24, 30, 37, 39, 77, 92, 98, 98, 114
E. Malaguti, M. Monaci, and P. Toth. A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20(2) :302, 2008.

[Marino *et al.*, 1999] cité page 82
A. Marino, A. Prugel-Bennett, and C.A. Glass. Improving graph colouring with linear programming and genetic algorithms. In *Proceedings of Eurogen99*, pages 113–118. University of Jyväskylä, Finland, 1999.

[Maturana, 2009] cité page 9
J. Maturana. *Contrôle générique de Paramètres pour les Algorithmes Évolutionnaires*. PhD thesis, University of Angers, France, 2009.

[Mehrotra and Trick, 1996] cité page 13
A. Mehrotra and M.A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4) :344–354, 1996.

[Méndez-Díaz and Zabala, 2006] cité page 13, 13
I. Méndez-Díaz and P. Zabala. A branch-and-cut algorithm for graph coloring. *Discrete Applied Mathematics*, 154(5) :826–847, 2006.

[Merz and Freisleben, 2000a] cité page 42
P. Merz and B. Freisleben. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, 4(4) :337–352, 2000.

[Merz and Freisleben, 2000b] cité page 42
P. Merz and B. Freisleben. Fitness landscapes, memetic algorithms, and greedy operators for graph bipartitioning. *Evolutionary Computation*, 8(1) :61–91, 2000.

[Merz, 2004] cité page 42, 42
P. Merz. Advanced fitness landscape analysis and the performance of memetic algorithms. *Evolutionary Computation*, 12(3) :303–325, 2004.

[Miller and Shaw, 1996] cité page 89, 89, 89, 90
B. L. Miller and M.J. Shaw. Genetic algorithms with dynamic niche sharing for multimodalfunction optimization. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 786–791, 1996.

[Morgenstern, 1996] cité page 14, 15, 15, 20, 30, 37, 49, 77, 79, 97, 98, 114
C. Morgenstern. Distributed coloration neighborhood search. In *Cliques, Coloring, and Satisfiability Second DIMACS Implementation Challenge* [Johnson and Trick, 1996], pages 335–358.

[Paquete and Stützle, 2002] cité page 14, 19
L. Paquete and T. Stützle. An experimental investigation of iterated local search for coloring graphs. In S. Cagnoni et al., editors, *EvoWorkshops*, volume 2279 of *LNCS*, pages 121–130. Springer, 2002.

[Park and Lee, 1996] cité page 13
T. Park and C.Y. Lee. Application of the graph coloring algorithm to the frequency assignment problem. *Journal of the Operations Research Society of Japan*, 39(2) :258–265, 1996.

[Plumettaz *et al.*, in press] cité page 14
M. Plumettaz, D. Schindl, and N. Zufferey. Ant Local Search and its efficient adaptation to graph colouring. *Journal of the Operational Research Society*, in press. doi :10.1057/jors.2009.27.

[Porumbel *et al.*, 2007a] cité page 3, 23
C.D. Porumbel, J.K. Hao, and P. Kuntz. Reinforced tabu search for graph coloring. *submitted paper*, 2007. revised in 2009.

[Porumbel *et al.*, 2007b] cité page 3
C.D. Porumbel, J.K. Hao, and P. Kuntz. A study of evaluation functions for the graph k-coloring problem. In *EA07*, volume 4926 of *LNCS*, pages 124–135, 2007.

[Porumbel *et al.*, 2009a] cité page 3, 75
C.D. Porumbel, J.K. Hao, and P. Kuntz. Diversity control and multi-parent recombination for evolutionary graph coloring algorithms. In *EvoCOP*, volume 5482 of *LNCS*, pages 121–132, 2009.

[Porumbel *et al.*, 2009b] cité page 3, 99
C.D. Porumbel, J.K. Hao, and P. Kuntz. An efficient algorithm for computing the partition distance. *submitted paper*, 2009.

[Porumbel *et al.*, 2009c] cité page 3, 75
C.D. Porumbel, J.K. Hao, and P. Kuntz. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *submitted paper*, 2009.

[Porumbel *et al.*, 2009d] cité page 2, 41
C.D. Porumbel, J.K. Hao, and P. Kuntz. Position guided tabu search for graph coloring. In *Selected Papers from the International Conference on Learning and Intelligent Optimization*, volume 5851 of *LNCS*, pages 148–162, 2009.

[Porumbel *et al.*, 2010] cité page 2, 41, 53, 72, 98
C.D. Porumbel, J.K. Hao, and P. Kuntz. A search space "cartography" for guiding graph coloring heuristics. *Computers & Operations Research*, 37 :769–778, 2010.

[Prestwich, 2002] cité page 20, 98, 114
S. Prestwich. Coloration neighbourhood search with forward checking. *Annals of Mathematics and Artificial Intelligence*, 34(4) :327–340, 2002.

[Radcliffe, 1994] cité page 76, 80
N.J. Radcliffe. The algebra of genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 10(4) :339–384, 1994.

[Rajagopalan *et al.*, 2008] cité page 88
R. Rajagopalan, C. K. Mohan, K. Mehrotra, and P. K. Varshney. EMOCA An Evolutionary multi-objective crowding algorithm. *Journal of Intelligent Systems*, 17(1/3) :107, 2008.

[Ramani *et al.*, 2006] cité page 13
A. Ramani, I.L. Markov, K.A. Sakallah, and F.A. Aloul. Breaking instance-independent symmetries in exact graph coloring. *Journal of Artificial Intelligence Research*, 26 :191–224, 2006.

[Reeves and Yamada, 1998] cité page 42
C.R. Reeves and T. Yamada. Genetic algorithms, path relinking, and the flowshop sequencing problem. *Evolutionary Computation*, 6(1) :45–60, 1998.

[Reeves, 1997] cité page 76
C.R. Reeves. Genetic algorithms for the operations researcher. *INFORMS Journal on Computing*, 9(3) :231–250, 1997.

[Régnier, 1983 et 1965] cité page 100
S. Régnier. Sur quelques aspects mathématiques des problèmes de classification automatique. *Mathématiques et Sciences Humaines*, 82 :20, 1983 et 1965. (reprint of *ICC Bulletin*, 4, 175- 191, Rome, 1965).

[Rodriguez-Tello *et al.*, 2008a] cité page 40
E. Rodriguez-Tello, J.K. Hao, and J. Torres-Jimenez. An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. *Computers & Operations Research*, 35(10) :3331–3346, 2008.

[Rodriguez-Tello *et al.*, 2008b] cité page 40
E. Rodriguez-Tello, J.K. Hao, and J. Torres-Jimenez. An improved simulated annealing algorithm for bandwidth minimization. *European Journal of Operational Research*, 185(3) :1319–1335, 2008.

[Russell and Norvig, 2002] cité page 1
S.J. Russell and P. Norvig. *Artificial Intelligence : A Modern Approach*. Prentice Hall, New Jersey, U.S.A., 2002.

[Ryan, 1995] cité page 42
J. Ryan. The depth and width of local minima in discrete solution spaces. *Discrete Applied Mathematics*, 56(1) :75–82, 1995.

[Sewell, 1996] cité page 13
E.C. Sewell. An improved algorithm for exact graph coloring. In *Cliques, Coloring, and Satisfiability Second DIMACS Implementation Challenge* [Johnson and Trick, 1996], pages 359–376.

[Shawe-Taylor and Žerovnik, 2001] cité page 14
J. Shawe-Taylor and J. Žerovnik. Ants and graph coloring. In *proceedings of the International Conference Artificial Neural Nets and Genetic Algorithms in Prague, Czech Republic*, pages 276–279, 2001.

[Shimodaira, 1997] cité page 88
H. Shimodaira. Dcga a diversity control oriented genetic algorithm. In *Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence*, pages 367–374, 1997.

[Smith *et al.*, 1993] cité page 89, 89, 89, 89, 90
R.E. Smith, S. Forrest, and A.S. Perelson. Searching for diverse, cooperative populations with genetic algorithms. *Evolutionary Computation*, 1(2) :127–149, 1993.

[Sörensen and Sevaux, 2006] cité page 88
K. Sörensen and M. Sevaux. Ma|pm memetic algorithms with population management. *Computers and Operations Research*, 33(5) :1214–1225, 2006.

[Stadler and Schnabl, 1992] cité page 42
P.F. Stadler and W. Schnabl. The landscape of the traveling salesman problem. *Physical Letters A*, 161(4) :337–344, 1992.

[Streeter and Smith, 2006] cité page 42, 42, 42
M.J. Streeter and S.F. Smith. How the landscape of random job shop scheduling instances depends on the ratio of jobs to machines. *Journal of Artificial Intelligence Research*, 26 :247–287, 2006.

[Tagawa *et al.*, 1999] cité page 82, 88
K. Tagawa, K. Kanesige, K. Inoue, and H. Haneda. Distance based hybrid genetic algorithm : an application for the graph coloring problem. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 2325–2332, 1999.

[Talbi and Weinberg, 2007] cité page 100
E.G. Talbi and B. Weinberg. Breaking the search space symmetry in partitioning problems an application to the graph coloring problem. *Theoretical Computer Science*, 378(1) :78–86, 2007.

[Tomassini *et al.*, 2008] cité page 43
M. Tomassini, S. Verel, and G. Ochoa. Complex-network analysis of combinatorial spaces : The NK landscape case. *Physical Review E*, 78(6) :66114, 2008.

[Trick and Yildiz, 2007] cité page 14
M. A. Trick and H. Yildiz. A large neighborhood search heuristic for graph coloring. In *CPAIOR 2007*, volume 4510 of *LNCS*, pages 346–360. Springer, 2007.

[Ursem, 2002] cité page 88, 88, 88
R. K Ursem. Diversity-guided evolutionary algorithms. In *PPSN VII*, volume 2439 of *LNCS*, pages 462–471. Springer, 2002.

[Voudouris and Tsang, 2003] cité page 8
C. Voudouris and E.P.K. Tsang. Guided local search. In F. Glover et al., editors, *Handbook of metaheuristics*, pages 185–218. Kluwer Academic Publishers, 2003.

[Weinberg, 2004] cité page 13
B. Weinberg. *Analyse et Résolution Approchées de Problèmes d'Optimisation Combinatoire : Application au Problème de Coloration de Graphe.* PhD thesis, Université de Lille 1, France, 2004.

[Welsh and Powell, 1967] cité page 13
D.J.A. Welsh and M.B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1) :85–86, 1967.

[Woo *et al.*, 1991] cité page 13
T.K. Woo, SYW Su, and R. Newman-Wolfe. Resource allocation in a dynamically partitionable bus network using a graph coloring algorithm. *IEEE Transactions on Communications*, 39(12) :1794–1801, 1991.

[Zdeborová and Krzakała, 2007] cité page 47, 47
L. Zdeborová and F. Krzakała. Phase transitions in the coloring of random graphs. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 76(3) :031131, 2007.

[Zhang, 2004] cité page 42
W. Zhang. Configuration landscape analysis and backbone guided local search. part i satisfiability and maximum satisfiability. *Artificial Intelligence*, 158(1) :1–26, 2004.

[Zhu, 2003] cité page 88, 88, 88
KQ Zhu. A diversity-controlling adaptive genetic algorithm for the vehicle routing problem with time windows. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, pages 176–183, 2003.

[Zufferey *et al.*, 2008] cité page 13
N. Zufferey, P. Amstutz, and P. Giaccari. Graph colouring approaches for a satellite range scheduling problem. *Journal of Scheduling*, 11(4) :263–277, 2008.

[Zufferey, 2002] cité page 13
N. Zufferey. *Heuristiques pour les problèmes de coloration des sommets d'un graphe et d'affectation de fréquences avec polarités*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2002.

# Publications

### Revues internationales avec comité de lecture

– Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, A Search Space "Cartography" for Guiding Graph Coloring Heuristics. *Computers & Operations Research*, vol. 37 (2010), pp. 769-778.

### Revues internationales avec comité de lecture (soumis ou en révision)

– Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, Reinforced Tabu Search for Graph Coloring.
– Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, An Efficient Algorithm for Computing the Partition Distance.
– Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, An Evolutionary Approach with Diversity Guarantee and Well-Informed Grouping Recombination for Graph Coloring.

### Conférences internationales avec actes *Lecture Notes in Computer Science*

– Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, Diversity Control and Multi-Parent Recombination for Evolutionary Graph Coloring Algorithms. Proceedings of Evocop 2009 (*9th European Conference on Evolutionary Computation in Combinatorial Optimisation*, Tübingen, Allemagne), LNCS 5482 : 121-132, Springer, 2009. [parmi les trois articles nominés pour le prix "best paper award"]
– Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, Position Guided Tabu Search for Graph Coloring. Selected papers from LION 3 (*Learning and Intelligent OptimizatioN* conference, Trento, Italie), LNCS 5851 : 148-162, Springer, 2009.
– Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, A study of evaluation functions for the graph K-coloring problem. Selected papers from EA'07 (*8th International Conference on Artificial Evolution*, Tours, France), LNCS 4926 : 124-135, Springer, 2007.

### Conférences et rencontres nationaux

- Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, Recherche Locale Guidée pour la Coloration de Graphes. *ROADEF 2009* – 10éme conférence de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, Nancy, France, 2009.
- Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, Un algorithme Guidé pour la Coloration de Graphe. Exposé à l'atelier *Journées Graphes et Algorithmes 09*, Sophia-Antipolis, Nice, France 2009.
- Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, Recherche Tabou renforcée pour la coloration de graphe. *ROADEF 2008* – 9éme conférence de la ROADEF, Clermont-Ferrand, France, 2008.

### Exposé invité

- Daniel Porumbel, Reinforced Tabu Search for Graph Coloring, Exposé à l'équipe d'optimisation ASAP (www.asap.cs.nott.ac.uk), Université de Nottingham, Royaume Uni, 2008.

# Algorithmes Heuristiques et Techniques d'Apprentissage – Applications au Problème de Coloration de Graphe –


## Résumé

Au cours des trois dernières décennies, les algorithmes heuristiques ont permis de réaliser des progrès remarquables dans la résolution des problèmes difficiles d'optimisation combinatoire. Cependant, la conception de ces algorithmes relève encore plusieurs challenges importants – en particulier, il semble qu'il est toujours difficile d'intégrer dans une heuristique une vue d'ensemble sur l'évolution de la recherche ou sur sa trajectoire. Prenant comme cadre expérimental le problème bien connu de la coloration de graphe, nous présentons de nouvelles stratégies qui font appel à certains mécanismes d'apprentissage pour rendre le processus de recherche plus "auto-conscient". Nous introduisons un algorithme qui est capable d'enregistrer sa trajectoire et d'interpréter sa propre évolution. Une analyse de l'espace de recherche a montré que les meilleures configurations visitées sont relativement proches les unes des autres, regroupées dans des *sphères* de rayon fixe. Avec ce type d'informations apprises, nous avons conçu : (i) des algorithmes de diversification qui "prennent garde" à ne pas visiter la même sphère à plusieurs reprises, (ii) des algorithmes d'intensification qui se focalisent sur l'exploration d'un périmètre limité en utilisant un parcours en largeur des sphères de ce périmètre, et (iii) des approches évolutionnistes pour gérer la diversité de sorte que les individus soient à la fois de bonne qualité eu égard à la fonction objectif et suffisamment distants les uns des autres. En fait, nous présentons une gamme de techniques (e.g. nouvelles fonctions d'évaluation) qui peuvent rendre la recherche heuristique "bien informée".



**Mots clés :** apprentissage et optimisation, enregistrement de la trajectoire de la recherche, intensification, diversité de la population, distance dans l'espace de recherche


# Heuristic Algorithms And Learning Techniques – Applications to the Graph Coloring Problem –


## Abstract

The last couple of decades have seen a surge of interest and sophistication in using heuristics to solve combinatorial optimization problems. However, the theoretical and practical research of these algorithms show there are many important challenges yet to be overcome – e.g. it seems that it is quite difficult to make a heuristic integrate a global vision over its own evolution or over its exploration path. Taking the well-known graph coloring problem as an experimental framework, we develop several new heuristics that integrate certain learning mechanisms so as to render the search process more "self-aware". For instance, we introduce an algorithm that is able to record its trajectory and to interpret its own evolution. A search space analysis showed that the best discovered potential solutions tend to occur relatively close to each other, clustered in spheres of fixed radius. Using such learned information, we developed : (i) diversification algorithms that "pay attention" not to visit the same sphere repeatedly, (ii) intensification algorithms assuring an in-depth exploration of a closed perimeter using a breath-first-search traversal of its spheres, or (iii) evolutionary algorithms that are able to keep the individuals sufficiently distant at all times, while not sacrificing population quality. In fact, we present numerous other techniques (e.g. new evaluation functions) that are able to render the heuristic search more "well-informed".



**Keywords :** search while learning, graph coloring, local search trajectory recording, intensification heuristics, population diversity, search space distance